N° d'Ordre : 2753 Année : 2000

THESE

En vue de l'obtention du grade de

DOCTEUR DE
L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

Discipline : Informatique

présentée par

Grégory BOURGUIN

Un support informatique à l'activité coopérative fondé sur la Théorie de l'Activité : le projet DARE

Directeur de Thèse : M. Alain DERYCKE

Jury

M. Vincent CORDONNIER , Président
M. Alain DERYCKE , Directeur
M. Bertrand DAVID , Rapporteur
M. William A. TURNER , Rapporteur
M. Bernard CARRE , Examinateur
M. Anders MORCH , Examinateur

Remerciements

Je tiens tout d'abord à remercier Monsieur Vincent Cordonnier, Professeur des Universités au Laboratoire d'Informatique Fondamentale de Lille (LIFL), pour m'avoir fait l'honneur d'accepter de présider le jury de ma soutenance de thèse.

Je tiens aussi à remercier Monsieur Bertrand David, Professeur des Universités au Laboratoire Interaction Collaborative, Téléformation, Téléactivités (ICTT) de l'Ecole centrale de Lyon, ainsi que Monsieur William A. Turner, Directeur de Recherche au CNRS, Laboratoire d'Informatique pour la Mécanique et les Sciences de l'Ingénieur (LIMSI), pour m'avoir fait l'honneur de rapporter et de juger mes travaux.

Un grand merci à Monsieur Bernard Carre, Maître de Conférences à l'EUDIL, et Monsieur Anders Morch, Dr. Scient. Associate Professor à l'Université de Bergen en Norvège, pour avoir accepté de faire partie de mon jury. Merci à Bernard pour m'avoir aidé et conseillé au cours de cette thèse. Merci pour m'avoir écouté parler de tâche, de rôle, de méta-niveau et de réflexivité pendant des heures. For Anders : Thank you for your work, your advice, your thesis, and for accepting to be a member of my jury.

Un immense merci à Monsieur Alain Derycke, Professeur des Universités au Laboratoire Trigone, et Directeur de ma Thèse. Merci pour m'avoir soutenu, aidé et conseillé, pour avoir cru en moi, et pour m'avoir permis d'être dirigé par quelqu'un que j'apprécie grandement.

Je tiens aussi à remercier tous les membres de l'équipe NOCE. En particulier, merci à Frédéric Hoogstoel pour m'avoir guidé, suivi, relu et toujours aidé lorsque j'en avais besoin. Merci à Xavier Lepallec et Yvan Peter et pour leurs relectures et leur amitié. Merci à Patricia pour m'avoir ouvert les portes des sciences humaines. Merci aux autres membres de l'équipe pour leur aide en diverses occasions.

Enfin, je tiens énormément à remercier ma compagne, Clotilde, ainsi que ma famille pour leur soutien et leurs encouragements. Clotilde, merci d'avoir relu l'ensemble de cette thèse et de m'avoir tant aidé pendant sa rédaction. Un immense merci à mes parents pour avoir cru en moi pendant toutes ces années et parce que, sans eux, sans leurs conseils, et sans leur amour, rien de tout ceci n'aurait pu arriver.

Table des Matières

Introduction

Le formidable développement des technologies informatiques, ainsi que la démocratisation de l'Internet laissent présager une nouvelle ère dans le domaine du travail : l'ère du Travail Coopératif Assisté par Ordinateur (TCAO). Avec le TCAO, chacun peut espérer un jour posséder un certain don d'ubiquité lui permettant d'interagir avec d'autres personnes disséminées au travers du monde, de manière synchrone ou asynchrone.

Malheureusement et aujourd'hui encore, le TCAO n'en est qu'à ses balbutiements. En effet, même si les technologies sous-jacentes, comme les réseaux informatiques ou encore les capacités de traitement des ordinateurs personnels, fournissent des bases technologiques solides au support d'applications puissantes de TCAO, il faut bien constater la faible acceptation des collecticiels par leurs utilisateurs potentiels. Ainsi, les chercheurs qui s'intéressent au domaine de recherche sur le TCAO tentent toujours de saisir l'essence de ce qu'est le travail coopératif, ceci dans le but de pouvoir enfin proposer aux utilisateurs des collecticiels répondant à leurs besoins.

Par nature, le TCAO se trouve au cœur de différents domaines de recherche. D'une manière générale, nous pouvons citer l'informatique, et les sciences humaines. La plupart des avancées dans le domaine de recherche du TCAO ont été fondées sur des résultats empiriques obtenus grâce à des interactions entre des informaticiens et des chercheurs en sciences humaines. Cette démarche pluridisciplinaire, s'intéressant de plus en plus à l'utilisateur final ainsi qu'à son contexte de travail, a permis de mettre en exergue le problème fondamental posé par les besoins de l'utilisateur qui émergent au cours de l'activité de travail coopératif. Ce constat a amené les concepteurs de collecticiels à recentrer le processus de conception des outils informatiques pour le TCAO sur les utilisateurs eux-mêmes, ainsi que sur les conditions environnementales de leurs activités.

Récemment, une nouvelle démarche de conception est apparue dans le domaine de recherche sur le TCAO. Celle-ci tente d'aller plus loin que l'étude ou la collecte des besoins d'une communauté d'utilisateurs particulière dans un contexte de travail particulier. Cette nouvelle approche tente de comprendre d'une manière générique les fondements du travail coopératif, ou de l'activité humaine, de manière à les rapprocher des fondements de l'informatique elle-même. En d'autres termes, il s'agit de créer des outils informatiques réellement fondés sur les concepts et mécanismes génériques fondamentaux des activités que nous tentons de supporter dans le TCAO. Cette approche semble très prometteuse, tout au moins si l'on se réfère à l'attention grandissante qu'elle obtient dans la recherche sur le TCAO. Malheureusement, les réalisations opérationnelles existantes construites selon cette démarche sont rares.

Cette thèse, effectuée au sein de l'équipe NOCE (Nouveaux Outils pour la Coopération et l'Education) du laboratoire Trigone, et sous la direction du Professeur des Universités, Monsieur Alain Derycke, propose les bases d'une nouvelle génération de collecticiels. Ces bases sont entièrement construites en adoptant la nouvelle démarche de recherche que nous venons d'exposer. Il est à noter que le laboratoire Trigone possède lui-même une structure pluridisciplinaire et se trouve intimement lié à l'institut CUEEP (Centre Université-Economie d'Education Permanente), spécialiste de la mise en place d'activités d'apprentissage coopératif. Ce contexte constitue un terrain idéal d'expérimentation et d'étude d'une certaine forme de TCAO : l'Apprentissage Coopératif Assisté par Ordinateur (ACAO).

Le chapitre premier de cette thèse présente comment nous sommes partis de l'analyse d'un environnement de TCAO développé au sein du laboratoire Trigone et utilisé dans de nombreux projets de recherches sur l'ACAO. Nous montrons comment cette analyse nous a amenés à chercher de nouvelles fondations pour la conception des environnements de TCAO, ainsi qu'à adopter cette nouvelle démarche qui tente de rapprocher les fondements des sciences humaines de ceux de l'informatique. Nous y définissons notre problématique générale et notre objectif qui consiste à réaliser un nouvel environnement générique de TCAO nommé DARE (Activités Distribuées dans un Environnement Réflexif).

Le chapitre deux présente l'introduction des sciences humaines dans le domaine de recherche sur le TCAO. Nous commençons par exposer l'approche classique de la conception de collecticiels pour terminer par l'étude des travaux de quelques pionniers qui ont adopté une démarche similaire à la nôtre. Ce chapitre conclue sur notre proposition pour la réalisation de DARE en tant qu'environnement de TCAO fondé sur les résultats de la Théorie de l'Activité.

Le chapitre trois présente en détail la Théorie de l'Activité du point de vue de la recherche sur le TCAO. Après un bref historique, et l'exposé de ses concepts et mécanismes de base, nous étudions les contributions de plusieurs chercheurs qui ont particulièrement influencé notre domaine de recherche en utilisant la Théorie de l'Activité. Nous concluons ce chapitre en présentant notre vision de ce que doit être un collecticiel, du point de vue de la théorie qui nous intéresse, et en formulant un ensemble d'enjeux à atteindre dans nos travaux. En particulier, ces enjeux préconisent de mettre en œuvre une très grande malléabilité dans les outils de support aux activités coopératives.

Le chapitre quatre présente ce qu'est la malléabilité et les divers travaux de recherche en informatique qui s'y rapportent. Nous identifions les mécanismes et moyens à mettre en œuvre pour que la malléabilité serve au mieux les besoins des utilisateurs. Nous introduisons notre approche tentant de lier au mieux les moyens de l'ingénierie logicielle permettant de supporter la malléabilité et les fondements de la Théorie de l'Activité.

Le chapitre cinq définit le modèle conceptuel de DARE. Nous y présentons les concepts et mécanismes fondamentaux de notre réalisation. Nous montrons en quelle mesure ces concepts et mécanismes sont intimement liés à ceux de la Théorie de l'Activité, de manière à maintenir la cohérence de notre approche.

Les chapitres six et sept présentent l'implémentation de DARE. Le chapitre six s'intéresse particulièrement au noyau réflexif du système, mêlant à la fois une architecture cadre qui propose un support générique aux activités coopératives, et l'approche de l'implémentation ouverte. Le chapitre sept focalise plus particulièrement sur l'architecture distribuée permettant aux utilisateurs d'interagir dans l'environnement de support aux activités coopératives proposé par DARE. Nous insistons dans ce chapitre sur la corrélation existante entre les moyens proposés, et les fondements de l'environnement.

Le chapitre huit constitue un bilan de nos travaux et propose des perspectives de travaux futurs. Nous présentons brièvement certains de ces travaux déjà entamés et basés sur nos résultats de recherche.

Pour terminer, nous concluons sur notre sentiment envers l'avenir du TCAO et sur l'idée d'un certain futur de l'informatique en général.

Chapitre 1

Le contexte

1.1 Le point de départ

Les travaux réalisés dans le cadre de cette thèse ont été amorcés suite à la thèse [52] de Frédéric Hoogstoel, aujourd'hui Maître de Conférence à l'EUDIL et chercheur au Laboratoire Trigone. Cette thèse, intitulée *« Une approche organisationnelle du travail coopératif assisté par ordinateur. Application au projet Co-Learn »*, s'intéresse au domaine du Travail Coopératif Assisté par Ordinateur (TCAO) et applique ses résultats dans le cadre particulier de l'Apprentissage Coopératif Assisté par Ordinateur (ACAO). De ce fait, les travaux d'Hoogstoel ont abouti à une architecture logicielle nommée ODESCA (Open Distributed Environment Supporting Co-operative Activities) dédiée au TCAO, et à un environnement d'ACAO créé grâce à ODESCA, nommé Co-Learn, et développé dans le cadre du projet du même nom [17][24][25]. Le point de départ de nos travaux a été l'étude d'ODESCA et l'observation de l'utilisation de Co-Learn au sein de l'institut CUEEP (Centre Université-Economie d'Education Permanente). Notre but était d'identifier les limites éventuelles de cette architecture logicielle, de manière à concevoir des environnements de TCAO encore mieux adaptés aux besoins de leurs utilisateurs.

1.2 Description d'ODESCA

1.2.1 Les enjeux sous-jacents

Suite à un large état de l'art du TCAO (mené il y a cinq ans), Hoogstoel identifie qu'un environnement de support à l'activité coopérative doit s'adresser à la fois à la coopération à court terme, et à la coopération à long terme. Il détermine alors six enjeux qui guident ses travaux. Les trois premiers enjeux apparaissent dès les activités à court terme : il s'agit de faciliter la communication, le partage de ressources et la coordination entre les membres du groupe de travail. Les trois enjeux suivants s'adressent plus spécifiquement à la coopération à long terme : il faut faciliter l'organisation du groupe, favoriser l'implication individuelle des membres et favoriser la cohésion du groupe.

De manière à atteindre ces enjeux, Hoogstoel définit qu'ODESCA doit permettre la mise en œuvre d'environnements globaux et intégrés de TCAO. Ce type d'environnement est à l'image de ceux préconisés dans le cadre des travaux du projet MOCCA [85][7] (travaux poursuivis dans le cadre du projet COMIC [18]). Un des apports principaux d'ODESCA est de répondre au problème de l'existence d'une *« multitude de logiciels d'aide à la coopération qui s'ignorent. Or chacun de ces collecticiels offre des fonctionnalités nécessaires mais insuffisantes pour assister une coopération durable au sein d'une organisation. L'idée est donc de créer un environnement d'accueil d'applications coopératives, comparable à un système d'exploitation distribué ouvert coopératif »* ([52], p 115). De ce fait, un *environnement intégré de TCAO* est défini par opposition aux autres collecticiels, que nous qualifierons de *collecticiels isolés*.

Sur ces bases, Hoogstoel conclue qu'ODESCA doit constituer un environnement virtuel global garant de la persistance et de la gestion de la coopération. Ce doit être un environnement d'intégration et de contextualisation capable de rendre coopératives des applications interactives isolées synchrones ou asynchrones. Pour répondre à ces objectifs, ODESCA fournit un modèle générique d'activité nommé MACAO [51] et une architecture cadre[1] supportant la régulation d'activités coopératives. De plus, ODESCA est défini comme un environnement malléable. La notion de malléabilité est inspirée des travaux menés dans le cadre de projets tels que *Object Lens* et *Oval* [80] [79] dont les auteurs mettent en avant la notion de malléabilité radicale. Très brièvement défini, un système malléable est un système

permettant à ses utilisateurs de l'adapter à leurs éventuels nouveaux besoins. Dans nos travaux, la malléabilité tient toujours une place très importante. Nous définissons cette notion, du point de vue de nos travaux et en détails, dans le chapitre 4 de cette thèse.

1.2.2 La structure organisationnelle

Etant donnée l'approche organisationnelle du TCAO adoptée par Hoogstoel, en cohérence avec les propositions du projet européen COMIC [18], ODESCA offre un support à l'émergence d'une organisation virtuelle. Une telle organisation présente un modèle hiérarchique à trois niveaux. Ces niveaux sont l'*organisation* qui contient une ou plusieurs *unités structurelles* contenant elles-mêmes une ou plusieurs *salles virtuelles*. La Figure 1 synthétise les différents niveaux de la structure organisationnelle d'ODESCA.

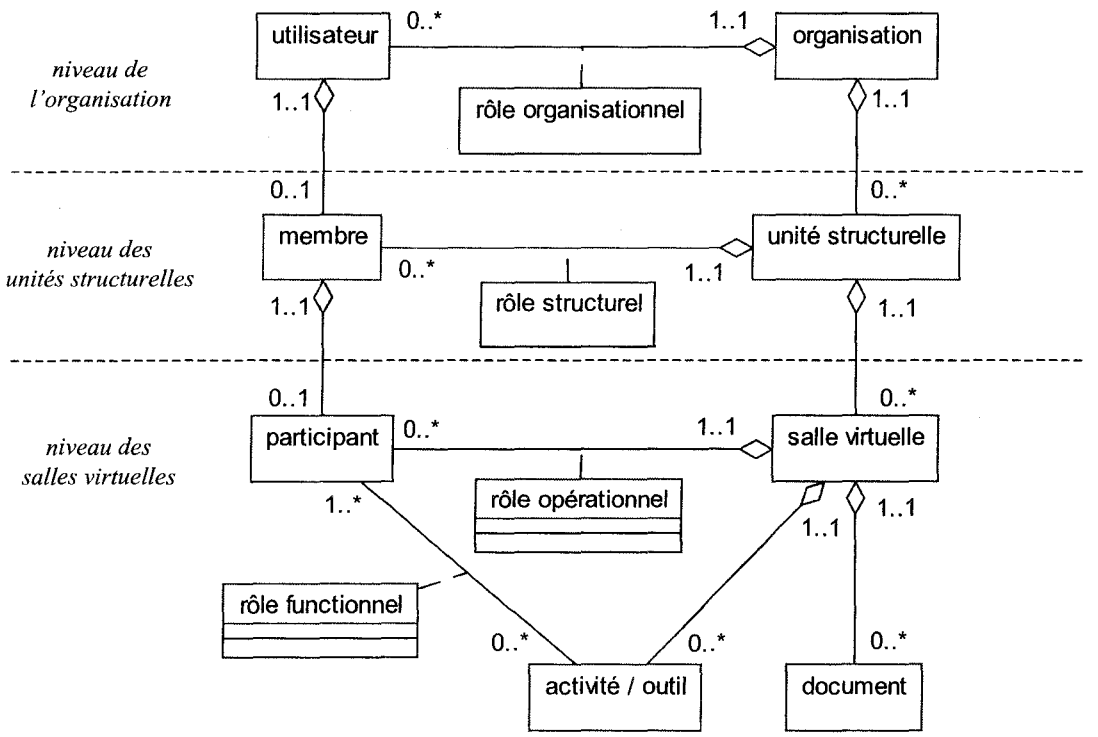

Figure 1. La structure organisationnelle à trois niveaux d'ODESCA (en notation type UML)

Au niveau de l'organisation, on trouve des *utilisateurs*. Chaque utilisateur joue un *rôle organisationnel*. Un tel rôle définit un ensemble de droits comme par exemple celui d'ajouter

[1] La notion d'*architecture cadre* réfère à la notion de *framework* en anglais. Une définition ainsi qu'une étude des architectures cadres peut être trouvée dans le chapitre 4 de cette thèse.

de nouveaux utilisateurs dans l'organisation. Chaque utilisateur peut être inscrit dans une ou plusieurs unités structurelles. Un utilisateur dans une unité structurelle est appelé *membre*. Chaque membre possède un *rôle structurel* qui permet par exemple d'ajouter d'autres membres dans l'unité. Un membre peut être inscrit dans une ou plusieurs salles virtuelles de son unité structurelle, auquel cas il devient un *participant* de la salle virtuelle. Chaque participant se voit attribuer un *rôle opérationnel* lui donnant des droits dans la salle virtuelle. Une salle virtuelle, en plus de ses participants, contient des documents ainsi que des outils. Les documents sont les documents partagés du groupe de participants. Les outils sont soit des outils individuels, soit des outils coopératifs. Lorsqu'un outil est coopératif, il correspond à un collecticiel isolé qui a été intégré dans l'organisation. Dans ODESCA, l'utilisation d'un tel outil est appelée *activité*. Dans une activité, un participant se voit attribuer un ou plusieurs *rôles fonctionnels*. Chaque rôle fonctionnel définit un ensemble évolutif de droits dans l'activité. Ces droits sont liés aux fonctions de l'outil-activité. L'évolution des droits est prédéterminée, codée dans les rôles fonctionnels, et s'effectue en fonction d'événements attendus dans l'activité.

Il est à noter que la notion de *rôle* prendra plusieurs sens dans cette thèse. Ici par exemple, cette notion est décrite du point de vue d'ODESCA. Elle se décline dans les différents types de rôles que nous venons de présenter. Nous verrons dans le chapitre 3 que la Théorie de l'Activité sur laquelle nous avons fondé nos travaux introduit une notion de rôle différente. Enfin, dans le chapitre 5, nous présenterons la notion de rôle telle que nous l'avons nous même (re)définie de manière à ce qu'elle synthétise l'évolution de notre conception d'un environnement de TCAO.

1.2.3 La malléabilité et l'ouverture

La malléabilité dans ODESCA est envisagée aux différents niveaux de la structure organisationnelle. Au niveau *organisation*, les utilisateurs peuvent changer leur propre rôle organisationnel ou celui des autres utilisateurs. Il leur est aussi possible de créer de nouvelles unités structurelles. Bien entendu, l'accès à cette malléabilité dépend des droits définis pour le rôle organisationnel courant de l'utilisateur. L'ensemble des rôles organisationnels est un ensemble fini. Chacun de ces rôles définit un ensemble de droits figés.

Au niveau d'une *unité structurelle*, et en fonction de leur rôle structurel, les membres peuvent changer leur propre rôle structurel, ou celui des autres. Ils peuvent aussi créer de nouvelles

salles virtuelles dans l'unité structurelle courante. L'ensemble des rôles structurels est un ensemble fini. Chacun de ces rôles définit un ensemble de droits figés.

Au niveau d'une *salle virtuelle*, les participants peuvent, en fonction de leur rôle opérationnel, changer leur propre rôle opérationnel ou celui des autres participants. Il est aussi possible de déterminer un ensemble d'outils accessible dans la salle virtuelle. Les outils sont choisis à partir d'une liste d'outils ayant préalablement été intégrés dans ODESCA. Les rôles fonctionnels de chaque participant peuvent être (ré)attribués. Enfin, la liste des documents partagés dans la salle virtuelle peut être (re)définie. L'ensemble des rôles opérationnels est un ensemble fini. Chacun de ces rôles définit un ensemble de droits figés. L'ensemble des rôles fonctionnels dépend des activités (outils coopératifs) intégrées dans la salle. Les droits des rôles fonctionnels peuvent évoluer en fonction d'un schéma d'évolution prédéterminé par le développeur de l'outil coopératif. Ces différents points définissent la malléabilité offerte aux utilisateurs, membres ou participants au sein d'un environnement de TCAO basé sur ODESCA.

Néanmoins, ODESCA prévoit une forme supplémentaire de malléabilité. Celle-ci propose la création de nouvelles activités à mettre en œuvre dans les salles virtuelles. Rappelons que dans ODESCA, une activité correspond à un outil coopératif associé à un ensemble de rôles fonctionnels. Cette forme de malléabilité est particulière car elle requiert l'intervention d'au moins un des développeurs d'ODESCA. Elle prévoit que les participants peuvent spécifier de nouveaux besoins envers une activité existante (la création d'un nouveau rôle fonctionnel par exemple), ou demander la création d'un nouveau type d'activité. Les besoins sont exprimés à partir d'un Modèle d'Activités Coopératives Assistées par Ordinateur (MACAO)[51] et utilisés par les développeurs pour intégrer de nouvelles activités dans l'environnement. Ces activités peuvent alors être utilisées au sein des salles virtuelles.

L'intégration de nouvelles activités dans l'environnement de TCAO est liée aux propriétés d'ouverture d'ODESCA. Il est en effet possible d'intégrer des collecticiels isolés existants et non-développés pour ODESCA. Les participants peuvent par exemple proposer l'intégration d'un outil *« tableau blanc partagé »* particulier qu'ils connaissent. Il est alors à la charge des développeurs d'ODESCA de trouver les points d'ouverture de cet outil (s'il en possède), de définir les rôles fonctionnels désirés (en les liant avec les fonctions ouvertes de l'outil), et de mettre en œuvre les moyens d'une intégration aussi fine que possible. Il faut par exemple définir comment l'outil s'exécute dans l'environnement global de TCAO. La tâche du développeur est facilitée par l'architecture cadre à objets offerte par ODESCA.

1.2.4 Le problème de l'activité

L'étude d'ODESCA et de son utilisation au sein de l'institut CUEEP nous ont permis d'identifier certaines limites introduites par son modèle conceptuel et le processus de conception des activités. Ces limites nous semblent fortement liées au concept d'activité sous-jacent.

Nous aimerions souligner que ces travaux d'étude ont été facilités car réalisés en collaboration avec des chercheurs spécialisés dans des disciplines différentes. Ces disciplines sont en particulier l'informatique, les sciences de l'éducation, la psychologie cognitive ou encore l'ergonomie. Une telle approche a été rendue possible du fait de la structure pluridisciplinaire de notre laboratoire d'accueil (Trigone), et de l'implication d'ODESCA dans divers projets tels que Co-Learn [17], MODEM [46][86] et plus récemment le projet Campus Virtuel ou « nouvelles interfaces pour le travail coopératif » [96] soutenu par le CNRS. Ce dernier n'est pas encore achevé et tente d'identifier les freins à la coopération qui peuvent exister dans les environnements de TCAO.

1.2.4.1 Un problème ontologique

Comme nous l'avons présenté ci-dessus, ODESCA a pour vocation d'être étendue par ses développeurs en intégrant de nouvelles activités spécifiées par les participants. ODESCA ayant principalement été utilisée dans la mise en œuvre d'environnements d'ACAO, les participants en contact avec les développeurs d'ODESCA sont des enseignants ou des concepteurs de dispositifs pédagogiques. Ceux-ci utilisent le concept d'unité structurelle pour créer des *cours* dans l'organisation. Le concept de salle virtuelle sert à définir des *salles de cours virtuelles* dans lesquelles interagissent des enseignants, des apprenants, des experts, etc. Malgré la bonne volonté des deux parties, les enseignants et les développeurs ont rarement réussi à coopérer de manière à définir de nouvelles activités. L'utilisation du concept d'activité dans la spécification des besoins a pour conséquence une incompréhension entre les différents protagonistes. Il apparaît clairement que les développeurs se réfèrent au concept d'activité défini dans ODESCA et MACAO (une activité est liée à un outil coopératif), alors que les enseignants considèrent l'activité à un niveau plus général. Pour ces derniers, l'activité englobe tous les outils de la salle virtuelle. Ainsi par exemple, les actions d'un participant sur un outil peuvent avoir un effet direct sur le mode d'utilisation d'un autre outil.

Ce type d'interactions ne peut être supporté dans ODESCA, en raison de son modèle conceptuel.

1.2.4.2 Un problème d'usages

Dans le cas où le problème de l'incompréhension a pu être résolu, survient un autre problème encore plus gênant. En tant que développeurs, nous sommes obligés de constater l'incapacité des participants à planifier totalement leurs activités. Or dans MACAO, tous les modes de coopération pouvant intervenir dans une activité doivent être spécifiés au cours de la phase de modélisation, de même qu'un nombre important d'autres informations (les types d'événements, etc.). Le système demande donc une planification fine et précise de l'activité, alors que les participants semblent préférer travailler avec une planification légère. Au cours de l'activité, ils adoptent un comportement opportuniste, adaptant leurs plans d'action à la situation. Dans l'ACAO en particulier, les enseignants assument et mettent en avant leur capacité à s'adapter aux progrès et réactions des apprenants. L'activité se révèle donc difficile à planifier. Ceci entre en contradiction, une fois encore, avec le concept d'activité d'ODESCA. L'idéal, face à ce problème, serait de permettre aux participants de re-spécifier une activité au cours de son exécution. Ceci serait d'autant plus intéressant lorsqu'il s'agit d'une activité à long terme. Malheureusement, dans la démarche adoptée pour ODESCA, de telles modifications ne peuvent être réalisées directement par les participants et dans l'activité. Elles sont à la charge des développeurs et réalisées dans un environnement de programmation externe. Pour atteindre cet objectif, il serait nécessaire d'augmenter considérablement la malléabilité du système.

1.2.4.3 Un problème structurel

Un autre point a été soulevé au cours de l'utilisation d'ODESCA dans l'ACAO. Les enseignants nous ont demandé un moyen de définir des sous-groupes dans une salle virtuelle, chaque sous-groupe possédant un ensemble d'outils et de documents particuliers. Etant donné la structure de l'organisation d'ODESCA, la notion de sous-groupe ne peut être que difficilement gérée. Il apparaît ici que la notion d'activité vue par les participants possède la propriété de se découper en sous-activités possédant chacune un ensemble de ressources particulières, mais conservant un lien avec la « sur-activité ».

Cette remarque nous amène à repenser la structure de l'organisation telle qu'elle existe dans ODESCA, ainsi que la place du concept d'activité. ODESCA définit des rôles fonctionnels, opérationnels, structurels et organisationnels. Or, le concept d'activité n'apparaît explicitement qu'au niveau des rôles fonctionnels. Pourtant, les rôles opérationnels, structurels et organisationnels sont les indicateurs d'une prise en compte inconsciente d'une activité à chacun de ces niveaux. Il devrait donc bien exister des activités coopératives de (re)définition de salles virtuelles, de création ou modification des unités structurelles et d'organisation. Dans ODESCA, chaque niveau ne met en œuvre qu'un ensemble fini et prédéterminé de rôles figés. Notre sentiment est que ces rôles, comme les rôles fonctionnels, devraient pouvoir évoluer. Il devrait aussi exister des outils permettant d'interagir d'une manière coopérative à chaque niveau. Par exemple, un outil de vote au niveau de l'organisation permettrait de décider ou non de la création d'une nouvelle unité structurelle. Ce schéma peut être répété dans tous les niveaux de l'organisation. Pour résumer, une unité structurelle apparaît comme une sous-activité d'une activité d'organisation. Une salle virtuelle peut être considérée comme une sous-activité d'une unité structurelle. Il ne semble exister aucune raison pour que la hiérarchie s'arrête à ce niveau.

1.2.4.4 Conclusion

Dans ODESCA, le concept d'activité est défini comme central. Cependant, plusieurs indicateurs nous conduisent à penser que toute l'essence de ce concept n'a pas été prise en compte. Il faut souligner que la notion d'*activité* d'ODESCA a été inspirée de celle définie dans AMIGO [52, pp. 120-122] [53]. Or, AMIGO fournit un modèle de l'activité centré sur la régulation des actions et interactions des participants. Il permet en particulier de concevoir des activités de communication de groupe. Il s'agit donc, non pas de l'activité au sens général du terme, mais d'un type d'activité. Le point de départ de nos travaux sera donc de revoir et d'étudier ce qu'est exactement l'*activité* de manière à en comprendre l'essence et à identifier la place qu'elle doit occuper dans un environnement de TCAO.

1.3 Les environnements de TCAO existants et basés sur ODESCA

L'étude de la mise en œuvre des différents environnements d'ACAO qui ont été créés sur les bases d'ODESCA nous a aussi permis d'identifier certains éléments qui ont guidé nos travaux. Nous avons d'ailleurs nous-mêmes participé à la création d'un de ces

environnements. Cette expérience a été d'une grande utilité pour formuler d'autres problèmes liés à la création d'un environnement de TCAO intégré.

1.3.1 Co-Learn [1992-1995]

La première réalisation qui ait été basée sur ODESCA est l'environnement d'ACAO baptisé *Co-Learn* [52][17]. Co-Learn utilise l'architecture logicielle fournie par ODESCA pour créer un environnement principalement basé sur des applications Windows au niveau des postes des utilisateurs. Certaines applications permettent, selon le modèle organisationnel décrit précédemment, de gérer l'organisation virtuelle à ses différents niveaux. D'autres applications correspondent aux parties clientes des activités dans les salles virtuelles. Ces dernières ont été créées pour interagir avec ODESCA et sont de ce fait très finement intégrées dans l'environnement. La Figure 2 est un exemple de salle virtuelle sur un poste client dans Co-Learn.

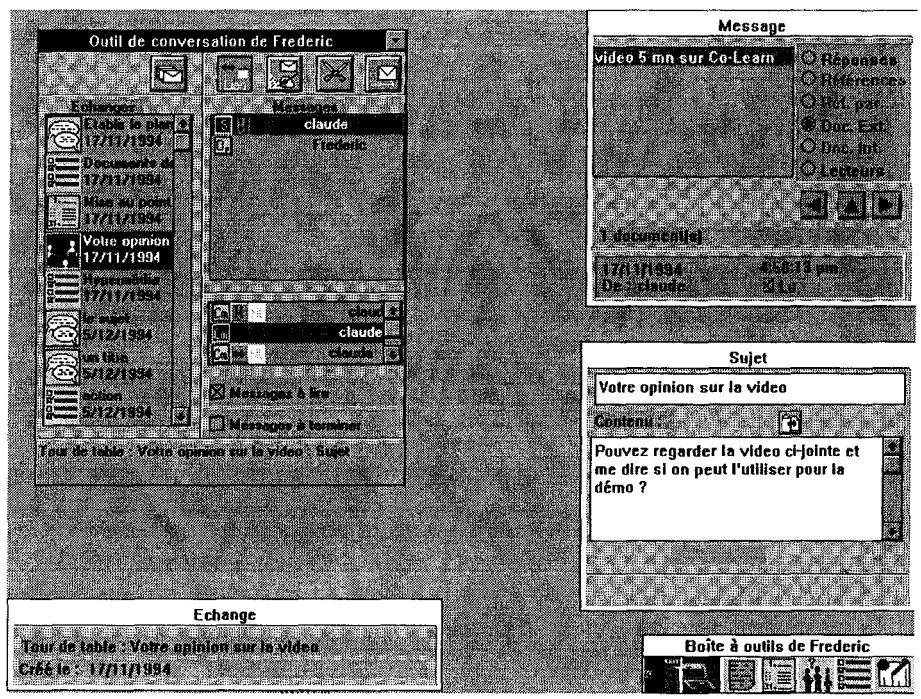

Figure 2. Un exemple de salle virtuelle sur un poste client dans Co-Learn

Un problème majeur de Co-Learn est qu'il met en œuvre des applications dédiées et développées pour fonctionner dans le système d'exploitation Microsoft Windows 3.11. L'utilisation de Co-Learn implique l'installation sur le poste client d'un ensemble non négligeable de programmes permettant d'interagir dans l'environnement d'ACAO. Ces programmes correspondent aux applications dédiées à l'administration de la structure organisationnelle, ainsi qu'aux activités pouvant apparaître dans une salle virtuelle.

Néanmoins, l'enjeu du support à la mobilité des utilisateurs a partiellement été pris en compte dans Co-Learn. En effet, chaque poste client ne doit pas être préalablement personnalisé pour qu'un utilisateur particulier puisse accéder à son environnement de travail. Le profil des utilisateurs est sauvegardé dans un serveur d'activités et chargé dans les applications clientes lors de la connexion. De plus, un mécanisme de mise à jour automatique a été mis en place de manière à ce que les nouvelles versions des outils puissent être chargées à partir du serveur.

Cependant, avec la montée en flèche d'Internet et sa démocratisation, il est de plus en plus aisé de trouver un ordinateur connecté au réseau. Ces postes clients possèdent généralement un navigateur Internet. Nous avons donc décidé d'augmenter le support à la mobilité des utilisateurs en créant un environnement d'ACAO accessible par Internet et ne nécessitant plus aucune installation préalable sur le poste client, si ce n'est celle du navigateur. Cet environnement a été nommé le *Campus Virtuel*.

1.3.2 Le Campus Virtuel [1996-1998]

Le *Campus Virtuel* [22][23][25] a été notre première participation à la réalisation d'un environnement de TCAO dans le cadre de cette thèse [50]. Le but du Campus Virtuel, comme celui de Co-Learn, est de créer un environnement global intégré d'ACAO. Le Campus Virtuel offre un accès aux activités d'enseignement au travers d'un simple navigateur Internet. Il se différencie donc principalement de Co-Learn en abandonnant l'utilisation d'applications Windows dédiées, favorisant ainsi la mobilité des utilisateurs. La Figure 3 représente un exemple d'environnement de travail d'un utilisateur dans le Campus Virtuel.

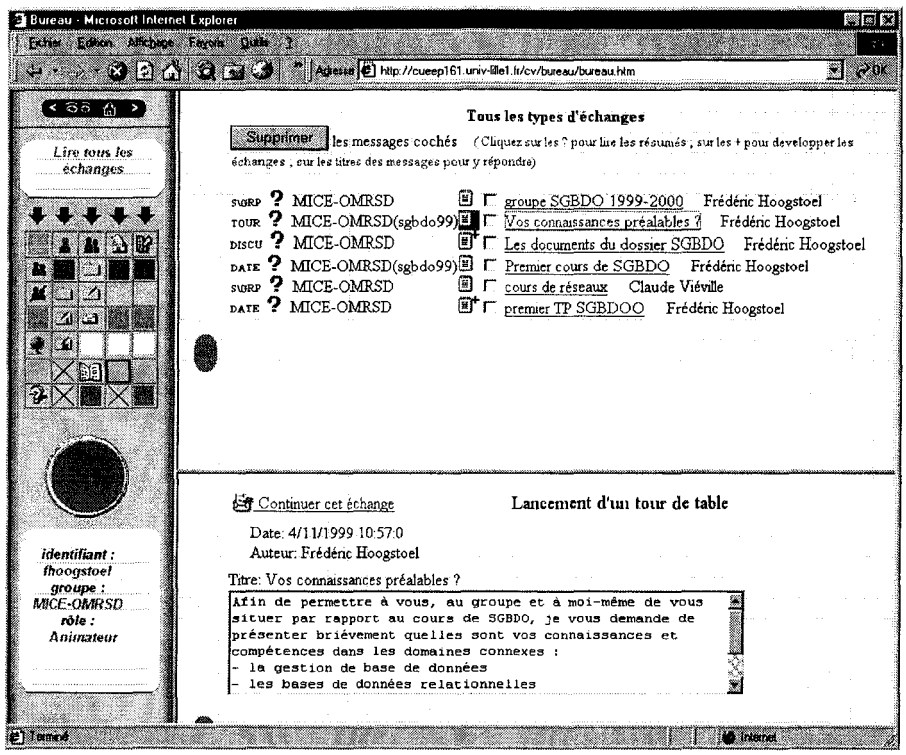

Figure 3. Un exemple d'environnement de travail dans le Campus Virtuel (2000)

Pour créer l'interface Internet d'ODESCA, nous avons mis en œuvre un mécanisme de génération dynamique de documents HTML (Hyper Text Mark-up Language) qui utilise des scripts CGI (Common Gateway Interface) écrits dans le langage Perl. Notons que la première version du Campus Virtuel date de 1996 et que CGI constituait à l'époque un des seuls moyens répondant à nos attentes. Le principe général de fonctionnement est assez simple. Un utilisateur s'identifie dans son navigateur au travers d'un formulaire. Il est alors reconnu au niveau du serveur HTTP (Hyper Text Transfer Protocol). Toutes les requêtes de l'utilisateur sont traitées par des scripts qui utilisent les informations contenues dans ODESCA pour générer une réponse personnalisée sous forme de document HTML. Ce mécanisme est synthétisé dans la Figure 4. La première version du Campus Virtuel a largement été utilisée dans le projet européen MODEM [86][46]. Une nouvelle version du Campus Virtuel est aujourd'hui en place et participe à d'autres projets comme [96]. Elle n'utilise plus de scripts CGI mais fonctionne dans un principe similaire en s'appuyant sur des technologies comme ASP (Active Server Page) [127]. Le Campus Virtuel est actuellement en cours d'industrialisation par la société Archimed.

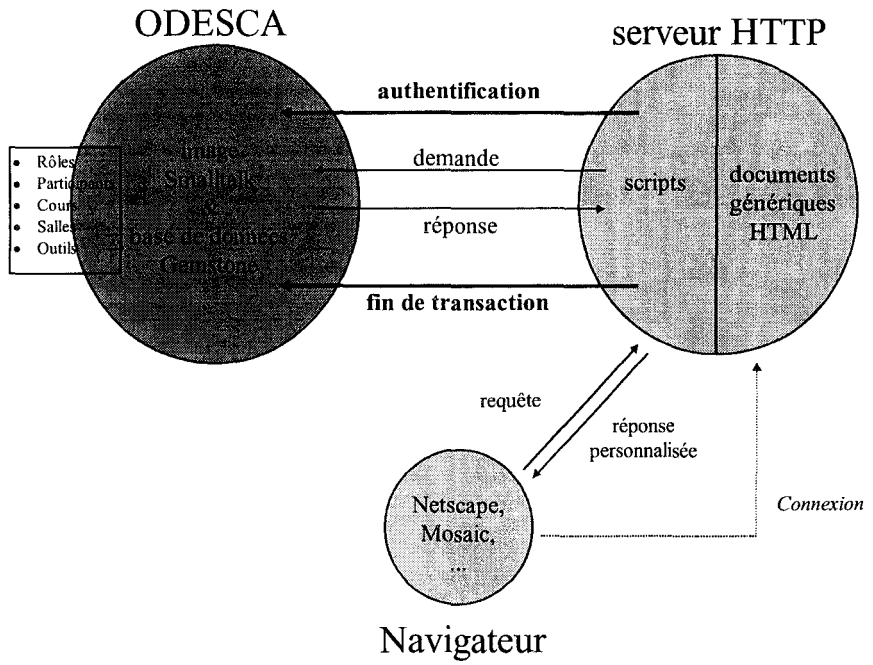

Figure 4. L'architecture du Campus Virtuel (1996)

Malheureusement, en passant de l'architecture de Co-Learn à celle du Campus Virtuel, nous avons pu remarquer la perte de propriétés importantes dans l'environnement d'ACAO. Les problèmes proviennent principalement du protocole de communication utilisé entre les clients et ODESCA. Ce protocole est incapable de maintenir des connexions symétriques pair à pair et ne supporte donc pas la notion de session. Autrement dit, les documents qui représentent l'environnement d'ACAO sur le poste client ne sont valides qu'au moment où ils sont générés. Une telle architecture ne peut supporter des activités synchrones. De simples documents HTML, même générés dynamiquement, ne permettent pas de mettre en œuvre la notion de co-présence souvent requise et bien connue du TCAO. C'est pourquoi le Campus Virtuel est principalement utilisé pour le support d'activités asynchrones [131][132].

Notons toutefois que nous avons pu, en tant que développeurs du Campus Virtuel, intégrer Microsoft Net Meeting pour en faire une activité synchrone d'ODESCA. Malheureusement, il s'agit plus d'une contextualisation que d'une réelle intégration. Net Meeting étant une application fermée, il est juste possible de proposer son démarrage à partir d'une salle virtuelle. La liste des utilisateurs connectés est créée à partir de la liste des participants présents dans la salle virtuelle. Une fois que Net Meeting est démarré, le Campus Virtuel n'a

plus aucun contrôle sur son exécution[1]. Enfin, il est impératif que les utilisateurs aient pré-installé Net Meeting sur leur poste de travail, ce qui revient à diminuer la mobilité que nous cherchions à supporter.

Malgré les problèmes non négligeables que nous venons d'exposer, nous pensons toujours que la mobilité des utilisateurs doit être supportée et que l'Internet est porteur d'une solution adéquate à ce problème. Le Campus Virtuel n'en est qu'une première résolution. Il nous faut donc travailler afin de définir une architecture logicielle basée sur Internet et répondant à toutes nos attentes.

1.4 Conclusion : la problématique générale

ODESCA peut être définie comme un *méta-collecticiel*. Ce terme, introduit par Ellis et Wainer [38], caractérise un système permettant de créer des collecticiels. A partir d'un modèle abstrait d'organisation virtuelle, ODESCA permet de créer des environnements globaux et intégrés de TCAO. Co-Learn et le Campus virtuel en sont deux exemples spécialisés dans le domaine de l'ACAO.

Toutefois, de par notre expérience dans la maintenance et la création d'un de ces environnements, nous avons pu identifier certains problèmes existants dans le modèle conceptuel d'ODESCA. Ces problèmes semblent intimement liés à la notion d'activité telle qu'elle y est utilisée. Nous pensons que la notion d'activité est importante pour bâtir les systèmes de TCAO, mais les résultats empiriques nous montrent que toute l'essence et les propriétés intrinsèques de l'activité n'ont pas été prises en compte. Ce manque se ressent jusque dans les fondements d'ODESCA et pose de nombreux problèmes dans l'utilisation des environnements de TCAO qu'elle supporte. C'est pourquoi, pour débuter nos travaux, nous avons décidé d'étudier ce qu'est l'activité. Une réponse peut être trouvée dans les sciences humaines. Nous espérons ainsi concevoir un méta-collecticiel qui permettra de créer des environnements de TCAO mieux adaptés aux besoins des utilisateurs. En particulier, l'étude de l'activité doit nous permettre de déterminer comment (et pourquoi il faut) fournir des environnements de TCAO beaucoup plus malléables.

Parallèlement, dans le but de poursuivre et d'améliorer la démarche amorcée lors du passage de Co-Learn au Campus Virtuel, nous déterminons plusieurs sous-objectifs principaux. Le

[1] Remarque : une version a été réalisée en utilisant des contrôles ActiveX. Cette réalisation a permis d'augmenter légèrement le contrôle de Campus Virtuel sur les instances de Netmeeting.

premier est de supporter la mobilité des utilisateurs. Le second provient du fait que nous ne voulions pas développer nous-mêmes de nouveaux collecticiels isolés, comme un tableau blanc partagé par exemple. Il nous faut donc proposer un environnement capable d'intégrer des collecticiels isolés existants, ceux-ci n'ayant pas forcément été développés pour cette intégration. Une vision utopique de cette démarche serait de permettre l'intégration de collecticiels découverts sur Internet, au fur et à mesure des besoins, dans un processus similaire à celui de la recherche d'informations.

Le Tableau 1 résume notre problématique générale. La réalisation que nous proposons dans le cadre de cette thèse pour atteindre ces objectifs est nommée DARE. DARE signifie *Activités Distribuées dans un Environnement Réflexif*. Nous montrerons dans cette thèse que DARE, à la différence d'ODESCA et suite à nos recherches, est plus qu'un méta-collecticiel : c'est un collecticiel réflexif.

Notre objectif général :
▪ supporter une certaine mobilité des utilisateurs.
▪ proposer un environnement intégrateur.
▪ proposer un environnement d'une grande malléabilité.
▪ saisir les fondements de l'activité humaine de manière à les faire converger avec les fondements d'un nouveau méta-collecticiel.

Tableau 1. Notre problématique générale

Chapitre 2

Les sciences humaines et le TCAO

Nous avons exposé dans le chapitre 1 que, dans le cadre de cette thèse, notre objectif est de proposer les fondements d'un nouveau genre de méta-collecticiel, ainsi qu'une réalisation nommée DARE (Activités Distribuées dans un Environnement Réflexif). Pour ce faire, nous tentons de saisir les fondements de l'activité humaine de manière à les faire converger avec les fondements de notre réalisation. L'étude de l'activité est un des thèmes principaux des recherches en sciences humaines.

Ce chapitre présente les liens de plus en plus forts qui unissent le domaine de recherche des sciences humaines et celui sur le Travail Coopératif Assisté par Ordinateur (TCAO). Après avoir exposé d'une manière générale les interactions qui existent entre ces deux domaines, nous présentons quelques travaux de recherche qui adoptent une démarche similaire à la nôtre, tentant de créer des relations plus fondamentales entre certaines théories des sciences humaines et la conception de systèmes de TCAO. Enfin, nous proposons notre nouvelle contribution fondée sur la Théorie de l'Activité.

2.1 L'introduction des sciences humaines

Les sciences humaines ont été introduites dans le domaine du Travail Coopératif Assisté par Ordinateur (TCAO) à partir des travaux de L. Suchman, et en particulier de son livre *"Plans as Situated Actions"* [126]. L'élément clé exposé par Suchman est qu'il ne peut exister de plans d'actions fixes et prédéfinis dans les activités humaines puisque les actions sont situées et correspondent à des comportements improvisés en réponse aux circonstances locales dans lesquelles les acteurs se trouvent. Cependant, l'auteur laisse une grande place aux plans d'actions et n'en rejette pas le concept. La réelle nuance se trouve dans le fait que ces plans d'actions, importants pour la réalisation du travail, doivent être révisables in situ. De ce fait, les approches largement utilisées à l'époque (1987) dans le domaine de recherche des Interactions Homme-Machine (IHM) et du TCAO, telles que celles tentant de modéliser les utilisateurs ou encore celles basées sur la planification, se trouvaient face à une large critique quant à leur viabilité. Les travaux de Suchman ont eu pour effet de recentrer la conception de systèmes de TCAO non seulement sur l'utilisateur, mais aussi sur sa situation de travail. La conséquence en a été l'introduction de certaines théories développées dans les sciences humaines en tant qu'outils conceptuels pour la conception de systèmes de TCAO.

2.2 Le problème du « great divide »

"Social scientists (primarily from sociology and anthropology), and computer scientists (primarily from software development, requirement engineering and artificial intelligence) have worked together on a variety of projects, seeking to create information systems more sensitive to human organisations and needs" [12].

Dans le but de répondre à ces nouveaux besoins, les chercheurs en sciences humaines ont souvent été intégrés aux groupes d'informaticiens créant ainsi des équipes pluridisciplinaires de conception. Ainsi, ces équipes hybrides présentent de nouvelles capacités non seulement dans la collecte d'informations relatives aux besoins des utilisateurs, mais aussi dans la critique de systèmes.

Cependant, dans de telles équipes, on peut assister à l'apparition d'une forte division du travail. Les chercheurs en sciences humaines sont généralement 'envoyés' sur le terrain dans le but d'étudier le travail en place et de formuler un ensemble de besoins qui doivent être pris en compte par le système qui doit le supporter. Les informaticiens se mettent alors au travail en utilisant leurs technologies et paradigmes pour générer l'application. Le système réalisé

subit ensuite les critiques fondées sur les sciences humaines qui avaient permis de le construire, il est modifié, puis critiqué à nouveau, et ainsi de suite jusqu'à ce qu'un équilibre s'installe ou que le projet soit abandonné. Ce type d'interactions au sein même des équipes pluridisciplinaires se trouve aujourd'hui placé au rang de méthode de travail reconnue et adoptée par toute la communauté de recherche en TCAO. Comme le soulignent Button et Dourish [35], pour être publié dans la littérature du TCAO aujourd'hui, il est de mise de rapporter un tel type de démarche scientifique. Cependant, la dichotomie générée par une telle division du travail ne semble pas la plus adéquate à la réalisation de systèmes directement opérationnels, ce qui expliquerait les nombreux allers et retours entre les phases de critiques et de re-conception.

De par nos propres expériences de travail entre chercheurs en sciences humaines et informaticiens dans un contexte favorable à la collaboration, nous avons pu vérifier que la réconciliation et la fusion des deux cultures est un problème difficile. Une des causes possibles semble être le manque de fondations réellement partagées. En effet, dans la démarche analyse/conception/critique/... exposée précédemment, le chercheur en sciences humaines n'a que peu d'informations sur les possibilités, limitations et implications des paradigmes utilisés par l'informaticien pour créer son système. De la même manière, pour l'informaticien, l'utilisation de théories des sciences humaines pour fonder sa conception est externe car uniquement révélée au travers de points spécifiques à implémenter, ou de lignes directrices à suivre.

Une solution émergeante dans la communauté scientifique du TCAO, et à laquelle nous adhérons, propose de créer des relations plus fondamentales entre les sciences humaines et celle de la conception de systèmes de TCAO. Il s'agit alors de rapprocher les fondements de ces deux disciplines, permettant ainsi de créer des méthodes de conception et systèmes opérationnels qui tiennent compte directement et explicitement des éléments et mécanismes fondamentaux des activités qui doivent être supportées.

2.3 Peu de réalisations opérationnelles

On constate que les nombreuses théories introduites par les sciences humaines sont encore aujourd'hui difficiles à utiliser pour réellement fonder les systèmes. Comme nous l'avons vu, leurs contributions n'apparaissent généralement que dans les premières et dernières étapes du processus de conception : respectivement l'analyse et la critique.

La recherche sur le TCAO s'interroge toujours quant à la manière de lier profondément les théories à la conception. Cette réflexion a d'ailleurs été le point central d'un récent workshop de la conférence ECSCW'99 [42] auquel nous avons participé et dont l'objet principal était de déterminer comment mieux fonder la conception des collecticiels en utilisant les théories telles que la théorie de la structuration, la théorie de la structuration adaptative, la cognition distribuée, les modèles situés, l'éthnométhodologie, ou encore la Théorie de l'Activité.

Depuis peu, quelques rares approches tentent de tirer partie des sciences humaines en introduisant leurs bases conceptuelles et leurs mécanismes dans les niveaux profonds du processus de conception. Notre contribution se situe exactement dans cette mouvance. Cependant, avant de décrire les fondements et la réalisation de DARE, notre environnement de support aux activités coopératives, nous souhaitons présenter quelques-uns des pionniers qui contribuent à l'heure actuelle à réduire la distance existante entre la conception des systèmes de TCAO et les théories qui les fondent.

2.4 Quelques pionniers

Il est important de comprendre en quelle mesure les travaux présentés ci-après se différencient de ceux qui ont contribué aux avancées du TCAO depuis une dizaine d'années. Comme nous l'avons exposé précédemment, l'introduction des sciences humaines n'est pas un fait récent.

Les nouvelles approches présentées ici apportent plus que des lignes directrices à suivre pour la réalisation des systèmes. Il s'agit pour certains d'un cadre méthodologique mettant directement en relation la théorie des sciences humaines sous-jacente et la conception. Pour d'autres, et à l'image de DARE, il s'agit de fournir une réalisation opérationnelle sous forme d'un prototype expérimental.

Même si ce type d'approche ne s'est pas encore généralisé au domaine de recherche, l'ensemble des contributions qui l'ont adopté ne se limite pas à celles qui sont présentées ci-dessous. Cependant, elles nous paraissent être les plus pertinentes dans le sens où elles nous ont particulièrement influencés et ont évolué parallèlement à nos travaux et avec certaines similitudes.

La première approche que nous considérons est celle qui a été développée dans le cadre du projet *wOrlds* et de son successeur *Orbit*, développés notamment à l'université de Queensland, et introduisant la théorie des *Locales*.

2.4.1 wOrlds, Orbit et les Locales

« [...] another iteration in the ongoing dialogue between the understanding of work and the design of systems to support work » [44]

Cette phrase résume bien l'approche des auteurs. Leur but, clairement exposé, est la définition d'une abstraction facilitant à la fois la compréhension du travail et la conception des systèmes qui le supportent. Leur démarche consiste alors à développer une théorie de l'activité collaborative basée sur la notion de *Locales*, et parallèlement, à développer des prototypes permettant d'évaluer et d'explorer leur théorie au travers d'implémentations.

Les prototypes développés pour vérifier cette théorie l'ont été dans le cadre de plusieurs projets. Le projet actuel nommé Orbit [81] constitue un environnement et une boîte à outil générique pour le support au TCAO. Orbit succède lui-même à un projet plus ancien nommé wOrlds qui fut le premier à implémenter le concept de *Locales*. Orbit, et wOrlds, d'un certain point de vue, sont très similaires. En effet, Orbit constitue une nouvelle implémentation de wOrlds, mise en chantier suite aux critiques concernant ce dernier [59]. Cependant, d'un projet à l'autre, il faut noter que la notion de locale sous-jacente n'a pas réellement évoluée et ce n'est qu'au niveau du passage à l'implémentation que les deux projets diffèrent.

La fondation des *Locales* provient du sociologue Strauss qui définit la notion de mondes sociaux : ce sont des groupes partageant un objectif commun et liés par les limites de la communication effective. A partir de cette définition, les auteurs considèrent que le rôle d'un environnement de TCAO est de fournir les moyens nécessaires aux interactions dans les mondes sociaux [130]. Une des notions clés de Strauss est qu'un monde social requiert un site (physique ou virtuel) et des moyens pour atteindre son objectif. L'agrégat du site et des moyens constitue une *Locale* [81].

« We define locale as the place that arises (i.e., that is created, maintained, and evolved) from the use of space and resources by a particular group of people or social world for their particular purpose. » [44].

A priori, cette définition semble correspondre aux buts que nous nous sommes fixés. Il s'agit bien de fournir aux groupes un environnement intégré supportant leur travail coopératif. En particulier, on peut souligner la mise en avant de la *création*, du *maintien* et de l'*évolution* des places de travail. Pourtant, il est difficile de définir exactement à qui les auteurs adressent ces tâches : aux concepteurs ou aux utilisateurs eux-mêmes ? Nous verrons plus tard que *notre*

approche insiste fortement sur la place prise par les utilisateurs dans la définition de leur propre environnement de TCAO.

L'étude des récentes et différentes versions d'Orbit permet de conclure clairement que l'utilisateur n'est pas en mesure de spécifier ou de faire évoluer profondément son environnement en utilisant Orbit lui-même. Pour ce faire, il doit passer par le concepteur du système. Pourtant, il s'agit d'un des buts que les auteurs se sont fixés à plus ou moins long terme. Pour ce faire, ils prévoient de réutiliser un sous-ensemble de travaux réalisés dans wOrlds. En effet, dans le cadre de ce projet, Tolone a défini un langage de spécification, Introspect, permettant la modification des locales [129] au cours de leur exécution. Grâce à celui-ci, Tolone tentait de fournir la flexibilité nécessaire au support à la collaboration, permettant d'adapter l'environnement à des pratiques de travail émergeantes et imprédictibles.

Introspect met en œuvre des mécanismes réflexifs en employant une architecture de méta-niveau. Cependant, cette réalisation focalise plus sur ses moyens techniques, que sur de réelles fondations en rapport avec la théorie des *Locales*. On peut d'ailleurs remarquer que les mécanismes réflexifs de wOrlds ne sont pratiquement jamais exposés du point de vue de l'utilisateur du système. En effet, la seule proposition en ce sens que nous ayons pu déceler consiste en la création d'une *Locale* particulière au sein de laquelle il est possible d'utiliser Introspect pour spécifier l'environnement. Nous voyons principalement trois limites à cette proposition.

La première limite est qu'Introspect ne nous paraît utilisable que par les développeurs du système lui-même. En effet, il nécessite d'entrer du code Smalltalk (le langage utilisé pour la réalisation), et donc de connaître finement le langage et les différents objets existant dans l'architecture de wOrlds (même si certains de ces objets correspondent à des abstractions existantes dans la définition d'une *Locale*). De ce point de vue, les possibilités d'adaptation de wOrlds sont proposées dans une démarche quelque peu similaire à celle existante dans ODESCA vue au chapitre 1. Ce sont les développeurs du système qui l'adaptent aux nouveaux besoins des utilisateurs.

La deuxième critique est que la spécification d'une *Locale* s'effectue dans une autre *Locale*. Ceci a pour conséquence, à notre avis, de décontextualiser la modification et de renforcer le sentiment de programmer l'environnement. La modification d'une *Locale* particulière concerne les éléments qui s'y trouvent. Il est plus aisé de re-spécifier des éléments dans le

contexte de leur utilisation, que dans un contexte externe. La dichotomie entre la *Locale* d'utilisation et la *Locale* de spécification est proche de celle existante entre une application informatique et son environnement de programmation. Il est souvent difficile d'abstraire les besoins apparus dans un contexte pour les retranscrire dans un autre, totalement différent.

La troisième et dernière critique majeure découle en partie des deux précédentes : même si l'utilisation d'Introspect se situe dans une *Locale* spécifique, l'utilisation de cet outil ne semble pas correspondre à une activité coopérative. Cet outil de spécification de l'environnement peut être mis à disposition d'un monde social, et donc d'un groupe, mais n'a pas été réalisé dans le but de mettre en œuvre des règles de médiation particulières au groupe, pouvant elles aussi évoluer, et qui définissent qui peut modifier quoi et à quel moment.

Pour terminer et appuyer les points précités, il est important de noter que dans Orbit, Introspect a pour l'instant disparu, ce qui semblerait souligner sa faible utilisabilité jusqu'à présent, du point de vue de l'utilisateur final. Les auteurs s'accordent d'ailleurs à conclure qu'il s'agit là d'un des plus grands challenges restant à relever dans ce projet [82]. Nous pensons que ces problèmes proviennent en partie de la théorie des *Locales* sous-jacente.

Les *Locales* ont la particularité de comporter cinq aspects [44]. Chacun de ces aspects caractérise la nature du travail selon une perspective différente. Nous n'allons pas exposer ici ces cinq aspects mais souligner qu'il est difficile d'identifier les mécanismes qui interviennent dans l'émergence de la coopération en particulier du point de vue de la communauté de pratiques. La théorie semble plus proche des soucis de réalisation du système insistant sur la notion de co-présence bien connue des collecticiels, de vues individualisées au niveau de l'interface utilisateur, ou encore d'articulation entre différents groupes de travail... Les notions faisant référence à une éventuelle co-construction ou expansion de l'environnement sont encore assez floues et d'ailleurs délaissées par les prototypes. Ce manque de clarté est certainement à l'origine des critiques que nous avons pu formuler précédemment. De plus, la définition d'une *Locale* laisse plus ou moins le champ libre à l'interprétation en ce qui concerne les éléments qui la composent. Ces éléments sont entre autres les objets, les outils et les ressources utilisées pour les interactions. Ces différents types d'éléments sont gérés différemment par le système. Ainsi dans wOrlds[1], les ressources correspondent à la vidéo et l'audio conférence. Celles-ci sont donc différenciées des outils de l'environnement (tout le monde possède les ressources, mais n'a pas obligatoirement accès aux outils). Pourtant,

[1] Dans Orbit, jusqu'à présent, la politique des droits d'accès aux éléments de la Locale sont très faibles : on est membre du groupe, ou on ne l'est pas. Des extensions sont annoncées par les auteurs pour de futures versions.

l'audioconférence et la vidéoconférence ne sont-ils pas des outils de communications ? Il n'est pas exclu d'imaginer que l'accès à ces outils (ressources pour wOrlds), soit différencié en fonction du rôle des participants. Ce type de différenciation entre les éléments qui composent une *Locale*, nous semble injustifié, et explique certainement pourquoi la spécification de l'environnement au travers d'Introspect n'a pas d'emblée été envisagée comme une activité coopérative à part entière, même si ce besoin avait été plus ou moins identifié.

Pour conclure, même si la démarche mise en œuvre ici semble similaire à la notre, elle présente néanmoins un grand nombre de divergences. En particulier, les points d'ancrage dans les sciences humaines existent bel et bien, mais la théorie des *Locales* a été utilisée et développée parallèlement dans le temps par des concepteurs peut être trop proches du système pour pouvoir extraire certaines dimensions importantes et intrinsèques à l'activité coopérative. De plus, nous pensons que la sociologie seule ne constitue pas une base suffisante à la compréhension du travail de groupe. Ces manques sont certainement à l'origine des critiques que nous avons exposés. Cependant, ils n'impliquent pour autant pas la condamnation d'une telle approche. En effet, selon les auteurs eux-mêmes, leur théorie ne demande qu'à évoluer suite aux expérimentations qui doivent être pratiquées grâce aux systèmes qui l'implémentent, comme les différentes versions d'Orbit. En ce qui nous concerne, nous ne nous croyons pas capable, en tant que concepteurs de systèmes et à cause de cette culture, de développer nous-mêmes une théorie de l'activité coopérative. Réciproquement, nous ne pensons pas que les chercheurs en sciences humaines soient capables de définir les fondements informatiques nécessaires à la conception de systèmes de support au TCAO. C'est pourquoi, nous pensons que la solution ne peut résider que dans une coopération profonde entre les acteurs de ces deux domaines de recherche.

2.4.2 Des fondements dans l'éthnométhodologie

Il serait difficile de passer outre la participation de l'éthnométhodologie à la recherche actuelle dans le domaine du TCAO. Selon Button, cet engouement pour la discipline serait plus ou moins accidentelle et découlerait du fait que les travaux de Suchman sont partiellement enracinés dans celle-ci. Quoi qu'il en soit, l'éthnométhodologie est aujourd'hui l'une des sciences humaines les plus saillantes dans notre domaine de recherche, notamment en Grande Bretagne.

« [...] as they [les chercheurs en IHM et TCAO] have taken on board Suchman's arguments, they also have taken on, perhaps unwillingly, an ethnomethodological influence. » [35]

Nous n'allons pas ici exposer en détail ce qu'est l'éthnométhodologie. Nous en serions d'ailleurs bien incapables n'ayant pas nous-mêmes utilisé ses fondements pour nos travaux. En une définition très simplifiée : l'éthnométhodologie étudie comment l'humain organise ses actions de tous les jours pour qu'elles soient compréhensibles par les autres dans les circonstances particulières de leur exécution. (Pour une définition plus détaillée, d'un point de vue TCAO, voir [35]).

Paul Dourish, à notre connaissance, est un des chercheurs en informatique les plus influencés par l'éthnométhodologie. L'ensemble de ses travaux, depuis au moins les huit dernières années, se rapportent plus ou moins à cette discipline. Même si la théorie sous-jacente diffère de celle que nous avons utilisé pour fonder DARE, il est intéressant d'étudier les apports de Dourish pour plusieurs raisons : la similitude entre les buts recherchés et les nôtres ; la similitude entre les solutions, en terme de stratégies logicielles, et les nôtres ; la place croissante prise par la théorie sociale à mesure de l'avancée de sa recherche. Ce tour d'horizon commencera par l'étude de Prospero, une boite à outil pour le TCAO, pour en arriver à la Technométhodologie, une nouvelle fondation au point de fusion entre l'éthnométhodologie et l'Informatique.

2.4.2.1 Prospero

Prospero est une boite à outils permettant de créer des applications de support au TCAO [34]. Le point d'ancrage de Prospero dans l'éthnométhodologie se révèle assez faible. Dourish ne l'utilise que pour argumenter en faveur de la flexibilité requise dans les systèmes de TCAO. Il souligne que, puisque les pratiques de travail apparaissent in situ, il est a priori impossible de connaître à l'avance les besoins de l'application qui les supportent. Il est clair que l'auteur est ici fortement influencé par les travaux de Suchman exposés précédemment. On peut alors se demander si Dourish, à cette époque, ne s'est pas lui-même laissé guider par l'engouement général existant autour de l'éthnométhodologie. En effet, d'autres théories, telle que la Théorie de l'Activité, posent le même postulat.

Une similitude majeure entre DARE et Prospero se trouve dans le but général du système. En effet, Prospero comme DARE peuvent être considérés comme des méta-collecticiels, c'est-à-dire des systèmes permettant de générer des collecticiels. Ils s'adressent donc tous deux à la

conception de systèmes en général et non pas à la conception d'un collecticiel particulier. L'objectif principal dans ce cas est de permettre de générer la plus grande gamme de collecticiels possibles. Prospero se réclame d'une flexibilité incomparable à celle offerte par les boites à outils traditionnelles. Pour ce faire, il met en œuvre des mécanismes de méta-niveau principalement basés sur l'Implémentation Ouverte (OI) proposée par Kiczales [63]. Dourish est alors un des (peut être 'le') premiers chercheurs à introduire la programmation réflexive [78] dans le TCAO [29]. Les mécanismes réflexifs de Prospero [30] permettent aux développeurs de le spécialiser in situ et de manière à répondre exactement aux besoins émergeants au moment de la création d'un collecticiel particulier pour un groupe particulier. Cette flexibilité de l'environnement s'avère être un point crucial car, selon les théories des sciences humaines comme l'éthnométhodologie ou la Théorie de l'Activité, il est impossible de connaître à l'avance les besoins de l'utilisateur.

Les utilisateurs de Prospero sont les développeurs d'applications coopératives. De notre point de vue, ceci introduit un paradoxe : les pratiques de travail émergent d'une activité à laquelle, typiquement, le développeur ne participe pas réellement. Dans cette approche, le dernier maillon de la chaîne capable de faire émerger le support à l'activité coopérative est celui qui développe l'application finale. Pour DARE, ce dernier maillon est l'utilisateur final ou plutôt, le groupe d'utilisateurs de l'application coopérative. Cette vision est toutefois quelque peu caricaturale. Dourish dans son analyse laisse entendre qu'il est possible au développeur utilisant Prospero de mettre en place des mécanismes dynamiques permettant à l'application générée de s'adapter en fonction de certaines conditions environnementales (le réseau informatique, etc.) ou laissant l'utilisateur spécialiser son environnement de travail. Quoi qu'il en soit et à l'inverse de DARE, Prospero n'est en aucun cas utilisable tel quel par un groupe d'utilisateurs voulant coopérativement spécifier ou faire évoluer son propre support à l'activité coopérative.

2.4.2.2 La Technométhodologie

La Technométhodologie [13] constitue la seconde facette des travaux de Dourish qui peuvent être directement mis en relation avec DARE. Plus précisément, la Technométhodologie est le fruit des travaux à la fois de Dourish, et de Button, un éthnométhodologue. Les deux chercheurs travaillent explicitement en tandem dans le but de rapprocher les fondements des sciences humaines et ceux de l'informatique.

A l'instar de nos travaux, la Technométhodologie traite de l'analyse et de la conception de systèmes en se basant sur une théorie provenant directement des sciences humaines. La Technométhodologie considère que la relation entre l'éthnométhodologie et la conception des systèmes doit dépasser le simple niveau pratique. Les interactions entre ces deux disciplines doivent prendre place au niveau des éléments qui fondent chacune d'elles. Ainsi, la Technométhodologie caractérise indiscutablement le nouveau type d'approche que nous tentons d'exposer. Cependant, si nous faisons abstraction de la théorie utilisée, elle présente deux différences principales avec nos travaux.

« This research is founded in a reevaluation of the role of abstractions in designing interactive systems [...] understanding how éthnomethodological understandings of human social action and interaction can be used, directly, in designing interactive technologies. » [35]

Premièrement, la contribution de la Technométhodologie ne se situe pas exactement au même niveau que la nôtre. Dans cette approche, il ne s'agit pas de créer un méta-collecticiel directement utilisable et permettant de générer de nouveaux collecticiels fondés sur la théorie utilisée, mais plutôt de concevoir un cadre méthodologique pour la conception de systèmes. Selon la Technométhodologie, au-delà même des abstractions concernant le domaine du travail dans lequel un logiciel est utilisé, il existe une forme de représentation commune à tout logiciel : les abstractions informatiques [33]. Le processus de conception ou encore la programmation elle-même sont des définitions, créations, et manipulations de telles représentations. La Technométhodologie est une nouvelle approche s'interrogeant, aux vues de l'éthnométhodologie, de l'essence même du rôle et de l'utilisation d'abstractions informatiques. Il n'existe pas de système mettant en œuvre la Technométhodologie. A l'inverse, DARE a pour but d'opérationnaliser informatiquement la création de systèmes de support à l'activité coopérative en se basant sur les concepts et mécanismes fondamentaux de la théorie des sciences humaines sous-jacente.

Secondement, l'intérêt de la Technométhodologie est porté à la conception de tout système interactif, incluant, mais ne se limitant pas aux systèmes de support au TCAO. Il s'agit en fait d'une généralisation du problème du TCAO lié à la notion de co-présence (*awareness*). Dourish définit la co-présence comme le fait de rendre l'activité de chacun visible pour les autres [32]. Il applique alors cette notion au cadre de l'activité d'un utilisateur seul, considérant l'ordinateur comme l'autre participant de cette activité. On voit ici fortement apparaître l'influence de l'éthnométhodologie. La question est : comment créer une situation

optimale pour que l'utilisateur soit à même de comprendre naturellement l'activité du système? Autrement dit, comment organiser chaque action du système pour que celle-ci soit compréhensible par l'utilisateur ? Une réponse partielle au problème est avancée, capitalisant principalement sur les concepts de base de l'éthnométhodologie et, une fois encore, sur l'utilisation de l'Implémentation Ouverte (OI) et de la programmation réflexive. Cependant, l'élargissement du problème posé éloigne la Technométhodologie de notre approche. Ici, l'ordinateur devient un des acteurs de l'activité. De notre point de vue, l'ordinateur, ou plus précisément le système destiné à supporter l'activité coopérative, n'est considéré que comme un des éléments médiateur de l'activité.

Pour conclure, il apparaît que la Technométhodologie n'en est encore qu'à ses balbutiements. Les publications s'y rapportant ne développent d'ailleurs principalement que la démarche générale. Seul un des éléments fondamentaux de l'éthnométhodologie a pour l'instant été pris en compte par les auteurs : l'*accountability*. Ce concept est très difficile à appréhender et est présenté dans [35]. Enfin, les apports de la Technométhodologie pour la conception de systèmes n'existent encore que sous la forme d'exemples particuliers utilisés pour exposer la démarche adoptée. Même si cette approche nous semble d'ores et déjà très intéressante, présentant un certain nombre de points communs avec la nôtre, il faudra, à notre avis, encore quelque temps avant de pouvoir en saisir tout l'intérêt.

2.5 Vers une nouvelle contribution des sciences humaines à la refondation des systèmes de TCAO

Les contributions précédentes ont été exposées dans le but de préciser la philosophie que nous avons développée et adoptée au cours de nos travaux dans le cadre de la conception de DARE. Un autre enjeu était de démarquer DARE en tant que pionnier dans la nouvelle génération des systèmes de TCAO. Comme nous l'avons souligné, et à la différence de la démarche adoptée dans wOrlds et Orbit, nous ne croyons pas en la définition d'une nouvelle théorie des sciences humaines créée par des concepteurs de systèmes, même si les racines de celle-ci se trouvent dans la sociologie. Nous ne croyons pas non plus qu'un lien fin avec une théorie établie, à l'image de celui existant entre Prospero et l'éthnométhodologie, soit suffisant pour réellement fonder la conception d'un système opérationnel de TCAO. D'un certain point de vue, nous nous sentons plus proche de la Technométhodologie, tentant de créer des relations profondes entre une théorie des sciences humaines et la conception des

systèmes. Cependant, la Technométhodologie et DARE divergent à la fois dans la contribution visée et dans la théorie des sciences humaines sous-jacente.

Paradoxalement et au-delà de la démarche générale tendant à fonder les systèmes dans les théories des sciences humaines, ces travaux présentent un certain nombre de points communs aux nôtres, en particulier au niveau des stratégies logicielles adoptées. En effet, à l'instar de wOrlds, Prospero et des propositions de la Technométhodologie, DARE repose lui aussi sur une architecture réflexive. Les techniques mises en avant pour sa réalisation s'inspirent de L'Implémentation Ouverte, de la programmation réflexive et du Meta-Object Protocol. Ces concepts seront développés plus loin dans cette thèse (cf. chapitre 6). Toutefois, l'introduction d'architectures réflexives semble déjà intimement liée à l'introduction des sciences humaines dans le TCAO et ceci quelle que soit la théorie utilisée pour fonder la conception.

Enfin, les auteurs de wOrlds, d'Orbit, et de la théorie des *Locales* présentent dans une récente publication [44] des approches similaires à la leur. Ils mentionnent en particulier la Théorie de l'Activité (AT) que nous présentons en détail dans le chapitre 3 et qui semble apporter une conceptualisation appropriée à la création d'une abstraction utile et commune à la compréhension du travail coopératif et à la conception des systèmes qui le supportent. Malgré cela, une question subsiste : comment opérationnaliser cette théorie ? Les auteurs précisent que ceci s'avère d'une grande difficulté car il s'agit alors d'utiliser l'AT dans un travail pour lequel elle n'a pas été créée. Il existe d'ailleurs peu de propositions opérationnelles basées sur l'AT, même si les chercheurs tentent toujours d'en apporter. Une proposition récente et pratique récemment publiée consiste en une liste de contrôle (checklist) basée sur l'AT et destinée à aider à la conception de systèmes [62]. Malgré sa réelle utilité, nous sommes encore loin d'un outil informatique générateur de collecticiels. Le même type de remarque a été formulé récemment par [8]. Celui-ci souligne que plusieurs efforts ont été faits dans le but de rendre l'AT opérationnelle, mais qu'aucune des propositions ne répond réellement au problème. En particulier l'ouvrage collectif *« Context and consciousness* : activity theory and Human-Computer Interaction *»* [94] avait pour but de remédier à cette absence de méthodes applicables et pratiques basées sur l'AT. Malheureusement, le livre ne contient que peu de contributions opérationnelles. Nous pouvons d'ores et déjà répondre que notre contribution, DARE[1], tente de relever ce défi.

La question légitime qui se pose alors est : pourquoi l'AT plus qu'une autre théorie ? Une première réponse, comme nous le verrons par la suite, réside dans l'objet de son étude

apportant une réponse à la vraie question en matière de support à l'activité coopérative : qu'est-ce que l'activité ? Plus précisément, pourquoi ne pas utiliser l'éthnométhodologie malgré sa forte influence dans le domaine de recherche qui nous intéresse ? Ce choix s'explique par plusieurs raisons : la première est l'héritage d'une certaine culture développée lors nos anciens travaux (exposés dans le Chapitre 1) qui introduisaient un concept d'activité proche de celui de l'AT ; la seconde raison est que l'AT considère dans son analyse à la fois la situation environnementale de l'activité et le sujet y participant. A la différence de l'éthnométhodologie qui focalise principalement sur la situation et les moyens à mettre en œuvre pour qu'elle soit compréhensible, l'AT donne une grande place aux mécanismes mentaux selon lesquels le sujet à la fois comprend et adapte son activité, ce qui correspond plus à nos attentes, en particulier dans le cadre de l'utilisation de DARE pour l'Apprentissage Coopératif Assisté par Ordinateur (ACAO) (cf. chapitre 8) ; enfin, nous croyons en la relative simplicité des modèles qu'AT définit, modèles qui nous semblent favorables au partage de représentations communes entre l'informaticien, le concepteur logiciel, le chercheur en sciences humaines et plus encore, la communauté des utilisateurs. C'est pourquoi, avant d'aller plus loin dans la description de DARE, il nous faut présenter cette théorie, ainsi que les différents travaux qui s'y rapportent, notamment ceux ayant une incidence sur notre nouvelle contribution.

[1] **dare.** *I.* n. défi m; **for a d.**, pour relever un défi. [49]

Chapitre 3

La Théorie de l'Activité

Comme nous l'avons présenté au chapitre 2, notre démarche tente de rapprocher les fondements de l'activité humaine, étudiés dans la Théorie de l'Activité, des fondements de la conception des environnements de TCAO. Dans un premier temps, ce chapitre présente la Théorie de l'Activité d'une manière générale. Puis nous présentons l'utilisation et les apports faits à cette théorie dans les domaines particuliers des Interactions Homme-Machine (IHM) et du Travail Coopératif Assisté par Ordinateur (TCAO). Cette étude nous permettra de donner notre définition d'un environnement de TCAO du point de vue de la Théorie de l'Activité, ainsi qu'une liste d'enjeux à atteindre dans nos travaux.

3.1 Histoire

La Théorie de l'Activité (Activity Theory ou AT) est un courant des sciences humaines qui profite d'une large audience dans le domaine de l'Interaction Homme-Machine (IHM) et du TCAO. Elle possède des fondations dans l'école historique et culturelle soviétique de la psychologie fondée par L. Vygotsky qui s'est principalement focalisé sur la médiation par le langage, alors que la Théorie de l'Activité se concentre sur la médiation par l'outil. Le père de

l'AT est A. Leont'ev, qui poursuivit les travaux de Vygotsky d'abord en tant qu'étudiant, puis en tant que collègue. Progressivement, l'AT s'est révélée être un corps de concepts dont le but est d'unifier la compréhension de l'activité humaine en fournissant les ponts vers les autres approches provenant des sciences humaines. En effet, les sciences du comportement ont toujours souffert d'une dichotomie existante entre l'individuel et le social. Si les sciences sociales utilisent le système social comme unité d'analyse, elles ont des difficultés à considérer l'humain lui-même. Les sciences du comportement, quant à elles, étudient les actions individuelles et ont des problèmes à en considérer le contexte. La solution offerte par la Théorie de l'Activité est la définition d'un concept intermédiaire, un contexte minimal pour l'étude des actions individuelles, qui sera considéré comme unité basique d'analyse : le concept d'activité. Elle offre un ensemble de perspectives sur l'activité humaine ainsi qu'un ensemble de concepts pour la décrire.

Plus précisément, l'AT est un cadre conceptuel pour étudier différentes formes de pratiques humaines en tant que processus développementaux, combinant à la fois et en même temps les niveaux individuels et sociaux. L'AT est caractérisée par la combinaison de perspectives sur l'activité humaine qui sont à la fois objectives, écologiques, et socioculturelles [60]. En d'autres termes, l'unité fondamentale d'analyse de l'AT est l'activité humaine qui est définie comme un système cohérent de processus mentaux internes, d'un comportement externe et de processus motivationnels qui sont combinés et dirigés pour réaliser des buts conscients. Cependant, et comme le souligne Kuutti [68], ce cadre est encore plus un agenda pour un programme de recherche qu'une théorie complète, même si les outils conceptuels qui y ont été développés semblent posséder de grandes qualités très prometteuses.

3.2 Les niveaux d'une activité

L'activité peut être divisée dans une structure hiérarchique comprenant 3 niveaux : l'activité, l'action, et l'opération. Cette structure, représentée dans la Figure 5, décrit l'activité (niveau le plus haut) comme étant réalisée au travers de chaînes d'actions (niveau intermédiaire) elles-mêmes réalisées au travers d'opérations (niveau le plus bas) – même si pour certains, d'un point de vue plus proche de la psychologie cognitive, les opérations peuvent elles-mêmes être divisées en blocs de fonctions [3, p.25].

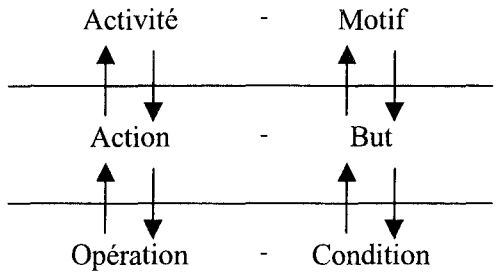

Figure 5. Les niveaux hiérarchiques d'une activité

3.2.1 Les actions

Les actions peuvent être individuelles ou collectives mais sont toujours dirigées vers un but conscient (intermédiaire par rapport à ce qui motive l'activité). Il est difficile, voire impossible, de comprendre une action en la considérant en dehors de son contexte, c'est-à-dire de son activité. Cependant, une action est souvent poly-motivée, c'est-à-dire qu'une seule et même action peut appartenir à plusieurs activités. De ce fait, l'action effectuée peut être interprétée différemment suivant l'activité dans laquelle on la considère. Ainsi, Kuutti [68] cite l'exemple d'une action concernant le rapport d'un projet. Celle-ci prend une connotation différente si on la considère dans le cadre de l'activité interne de management du projet, ou dans le cadre d'une activité de compétition pour une promotion.

L'exécution d'une action est planifiée, en utilisant un modèle, et au cours d'une phase appelée *orientation*. Sa réalisation peut donc être considérée comme une séquence *orientation - exécution*. Plus le modèle mis en œuvre est fiable, plus l'action a des chances d'atteindre son but. Dans le cas contraire, le sujet entre dans une phase d'analyse et d'apprentissage au cours de laquelle il sera certainement amené à modifier le modèle défaillant et, ainsi, augmenter son expérience. Ceci réfère au concept de Plan Interne d'Action (*Internal Plan of Action ou IPA)* développé dans l'AT. L'IPA est proche des concepts cognitifs de mémoire de travail et de modèle mental, à la différence près qu'il ne renvoie pas à des modèles spécifiques, mais plutôt à la capacité générale de les créer et de les transformer. L'IPA représente la capacité humaine d'exécuter des manipulations sur des représentations internes d'objets externes, avant d'exécuter ces actions, sur ces objets, et dans la réalité [1]. Autrement dit, la fonction de l'IPA est de simuler les résultats futurs de possibles événements avant d'exécuter les actions dans la réalité.

3.2.2 Les opérations

Les opérations sont exécutées inconsciemment et sont orientées par une base d'orientation non consciente. Cette base d'orientation est établie au travers de l'expérience apparue au contact des conditions matérielles concrètes de l'opération [1].

Les opérations correspondent à des actions dont le modèle s'est démontré réellement fiable en fonction de certaines conditions. On peut considérer une opération comme une action qui est descendue d'un niveau dans la structure hiérarchique de l'activité, du fait de la quasi-disparition de sa phase d'orientation. Elle est alors exécutée plus rapidement et répond, de façon inconsciente, à des conditions spécifiques. Les actions opérationnalisées sont aptes à participer à la création de nouvelles et plus amples actions. L'exemple le plus souvent cité est celui de l'utilisation d'une boite de vitesse qui, au départ, est difficile et lente car réfléchie et planifiée. Peu à peu, celle-ci devient fluide et inconsciente, apte à participer à des actions de plus haut niveau (ex. ralentir). Les opérations nous permettent d'agir sans penser consciemment à chaque petit pas d'exécution. D'ailleurs, et comme le notent Raeithel et Velichkovsky [108], la cause d'une erreur commise dans l'exécution d'une action constituée d'opérations bien maîtrisées est difficile à capturer car ces opérations sont réalisées de façon inconsciente (ce qui entraîne par exemple des questions telles que : « Ai-je bien fermé la porte à clef en partant ? »). Les opérations ne sont pas accessibles à l'auto-réflexion consciente de l'acteur.

Il arrive souvent, lorsque les conditions d'exécution d'une opération changent, que l'opération retourne au niveau de l'action, demandant une ré-analyse de la situation et une nouvelle phase d'orientation. Ceci arrive par exemple lorsque l'on descend un escalier dont les marches sont trop serrées, ce qui oblige l'opération de descente à revenir au niveau d'une action consciente, demandant plus d'attention, et exécutée plus lentement.

Dans la vie, il n'existe pas une correspondance un à un entre un but à atteindre et la manière d'y arriver [61]. Ceci signifie qu'il n'y a pas unicité entre une action et l'ensemble des opérations qui participent à sa réalisation. Selon l'AT, le déclenchement des opérations n'est pas uniquement guidé par les conditions environnantes. Elles sont en fait déterminées par la structure générale de l'action dans laquelle elles ont été incorporées. Ceci permet par exemple de représenter comment un sujet peut apprendre à contrôler ses réactions immédiates face à certaines situations. Enfin, comme le souligne Nardi [95], l'exécution d'opérations qui ne

réalisent pas d'action orientée vers un but peut être comparée à l'opération d'une machine qui aurait échappé au contrôle de l'humain.

3.2.3 L'activité

Comme le décrit Kuutti [68] en se référant à Davidov, Zinchenko, et Talyzina [19], une action peut elle-même monter d'un niveau et devenir une activité. Réciproquement, une activité peut correspondre à une action dans une activité d'un niveau plus général. Ainsi, dans l'AT, les limites entre les différents niveaux de la structure hiérarchique d'une activité sont mouvantes et floues. Ceci est particulièrement remarquable entre les niveaux activité et action car la classification dépend totalement du point de vue adopté.

La dynamique action-opération et la possibilité de construire, au fil du temps, des actions de plus haut niveau à partir d'actions opérationnalisées constitue une propriété fondamentale du développement humain. Ces mécanismes nous paraissent importants car ils viennent contribuer à la formation de l'expérience des sujets. Nous verrons plus loin que cette expérience joue un rôle important dans le développement de l'activité et la création des éléments qui la composent.

Finalement, nous pouvons dès à présent remarquer la manifestation d'un phénomène réflexif au sein de l'activité. En effet, nous avons vu que les opérations dépendent fortement du contexte, de la situation d'exécution des actions auxquelles elles participent. Or les actions ont tendance à transformer cette situation. Ainsi, l'activité apparaît comme continuellement influencée par une situation qu'elle ne cesse de modifier.

3.3 Les propriétés de l'activité

La Théorie de l'Activité définit huit propriétés basiques du concept d'activité :

1) Une activité possède un objet (matériel brut ou espace problème) vers lequel elle est dirigée, qui la différencie d'une autre activité, et qui en motive l'existence. Derrière l'objet, se trouve toujours un besoin ou désir auquel l'activité doit répondre [95]. On parle d'ailleurs parfois de motivation 'objectifiée' ou *« objectified motive »* [15].
2) Une activité est un phénomène collectif.
3) Une activité possède au moins un sujet (acteur) *actif* qui comprend et est en accord avec son objet. Tous les participants d'une activité ne comprennent pas ou ne reconnaissent pas

forcément l'existence de l'objet de l'activité, auquel cas ils sont identifiés comme des sujets *passifs*. Le sujet peut être individuel ou collectif.

4) Une activité est réalisée par ses participants au travers d'actions orientées vers des buts conscients.
5) Une activité existe dans un environnement matériel qu'elle transforme.
6) Une activité est un phénomène de développement historique.
7) Les contradictions[1] qui y apparaissent en sont la force de développement.
8) Les relations existantes entre les éléments d'une activité sont médiatisées culturellement.

L'énoncé de ces propriétés, dans une première approche, nous permet de considérer des apports intéressants pour le TCAO. En particulier, l'activité est directement considérée comme un phénomène collectif[2], dirigée vers un objet idéal qui la motive et autour duquel est constitué un groupe de sujets interagissant jusqu'à sa réalisation. Notre but étant de fournir un support à l'activité coopérative, certains de ces points ont déjà une résonance particulière pour notre conception : l'activité transforme son environnement, c'est un phénomène dynamique, de développement historique et empli de contradictions. Ces propriétés résument les notions générales qui nous ont mené vers une approche nouvelle pour fonder la conception de notre système. Pour expliquer et comprendre plus en profondeur les concepts et mécanismes sur lesquels reposent ces propriétés, nous allons exposer les travaux d'Engeström (et son modèle de la structure basique d'une activité), ainsi que ceux de Kuutti grâce auxquels l'AT a été identifiée comme une réponse intéressante aux problèmes des IHM et du TCAO.

3.4 La structure basique d'une activité

3.4.1 La médiation par l'outil

Engeström [39][40] a défini un modèle structurel simple du concept d'activité exprimant la médiation existante entre le *sujet* et l'*objet* de l'activité. Cette médiation est représentée par le concept d'*outil* représentant tout ce qui est utilisé dans le processus de transformation, incluant aussi bien les outils matériels que les outils pour penser. Ce modèle est représenté Figure 6.

[1] Ces sont les contradictions entre les différents éléments qui la constituent. Ceux-ci sont détaillés dans la structure basique d'une activité d'Engeström (cf. 3.4.2 et 3.4.3)

[2] Le collectif considéré ici ne se limite pas forcément à la notion de groupe de travail généralement considéré dans le TCAO. En effet, l'AT donne une place importante aux interactions entre des sujets impliqués dans plusieurs et différentes activités, souvent contradictoires et génératrices de conflits.

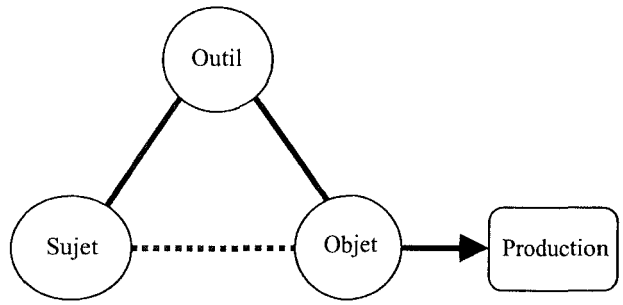

Figure 6. La relation médiatisée au niveau individuel

L'outil influence fortement l'activité. En effet, l'outil à la fois permet et limite : il permet au sujet de réaliser l'objet de son activité, mais limite en masquant une partie du potentiel de transformation de cet objet. Les outils supportent et complémententent les capacités humaines dans la construction de systèmes plus efficaces, permettant des accomplissements plus hauts. En contrepartie, l'outil participe à la formation des objectifs des sujets qui l'utilisent car il porte en lui des buts implicites qui y ont été mis par ses développeurs. L'utilisation d'un outil particulier change la structure de l'activité et peut même résulter en de nouveaux buts à satisfaire. D'ailleurs, Ellen Christiansen [15] rappelle qu'un outil doit servir le double but de permettre de *faire* quelque chose et de *rappeler* que quelque chose doit être fait. D'un autre côté, l'outil est lui-même transformé et construit au cours de l'activité. Il est souvent modifié ou adapté par les sujets, en fonction de leurs besoins émergeants, de leurs buts, et de leur expérience. Ainsi, cette expérience (cf. paragraphe 3.2) se retrouve dans l'outil qui porte en lui l'héritage culturel de la situation. La médiation par l'outil correspond alors à un moyen de transmettre une certaine culture. Il est important de noter que, si l'outil ne possède pas les propriétés adéquates à une adaptation potentielle, il peut être définitivement rejeté de l'activité.

On voit ici réapparaître une certaine réflexivité. D'une part, l'intégration d'un outil dans une activité a pour conséquence d'en influencer le cours, mais d'autre part, cette activité a tendance à le modifier. En d'autres termes, et comme nous l'avons déjà souligné, les actions des sujets au cours de l'activité ont tendance à créer une situation qui, elle-même, influence l'activité et donc les actions futures.

Dans l'AT, l'intégration de nouveaux outils est posée en terme de création d'organes fonctionnels ou *« functional organ »* [134]. Ces organes fonctionnels correspondent à des outils externes qui sont intégrés et expérimentés comme des propriétés de l'individu.

L'exemple souvent cité est celui de l'œil humain qui, équipé de lunettes, compose un organe fonctionnel permettant une meilleure vision. Kaptelinin [61] utilise la notion d'organe fonctionnel pour reposer le problème central des IHM. Selon lui, il s'agirait de trouver le moyen optimal permettant l'intégration d'outils informatiques dans la structure de l'activité humaine.

3.4.2 Généralisation

Le modèle de la relation médiatisée au niveau individuel (cf. Figure 6), tel qu'il a été décrit, ne permet pas de considérer toutes les relations qui existent entre un individu et le reste de son environnement, en particulier et comme nous l'avons vu, le fait que l'activité soit un phénomène collectif. Il a donc été étendu par Engeström pour représenter la *structure basique d'une activité* qu'il définit comme le modèle systémique le plus simple en terme d'unité d'analyse. De nouveaux éléments viennent ainsi se greffer sur ceux de l'activité individuelle, de même que de nouvelles relations entre ces éléments.

L'analyse est étendue pour montrer que l'individu n'est pas isolé mais fait partie d'une *communauté*. La communauté représente l'ensemble des sujets qui partagent le même objet d'activité. Puisque l'activité est médiatisée, les relations qui existent entre ses divers éléments sont elles-mêmes médiatisées par de nouveaux concepts contenant, eux aussi, l'héritage culturel de la situation. Raeithel et Velichkovsky [108] soulignent que, lorsque de nouveaux membres arrivent dans une communauté de pratiques, ils doivent s'approprier une connaissance distribuée et située du pourquoi, quoi et comment des actions typiques de celle-ci. On voit ainsi apparaître deux nouveaux éléments : le concept de *règles* (*rules*) qui médiatise principalement la relation sujet–communauté et celui de *division du travail* (*division of labour*) qui médiatise la relation objet–communauté.

Au cours de l'activité, les règles peuvent être plus ou moins explicites et contrôlent les relations entre le sujet et la communauté. Elles représentent des normes, des conventions, des procédures administratives, des pratiques de travail acceptées, ou encore des relations sociales. En d'autres termes, les règles définissent ce que signifie être membre d'une communauté. Comme tout élément médiateur, les règles cristallisent en elles un certain héritage culturel relatif à la relation qu'elles médiatisent. Par exemple, il est aisé de comprendre comment des lois sont construites et modifiées au cours du temps en tenant compte de l'expérience des sujets qui les ont créées. Cet exemple souligne d'ailleurs l'un des points les plus importants de l'AT rapporté par Kaptelinin [60] : la nature de tout artefact

médiateur ne peut être comprise que dans le contexte de l'activité humaine. Chacun sait qu'il est souvent difficile de comprendre certaines lois si l'on ne considère pas le contexte dans lequel elles ont été créées. Enfin, en tant qu'élément médiateur, les règles sont ouvertes à de nouveaux développements au sein de la communauté, modifications qui viendront compléter l'héritage culturel qu'elles contiennent.

La division du travail correspond à l'organisation de la communauté dans le but de la réalisation de l'objet. Elle définit ce que chaque sujet d'une communauté doit faire pour réaliser l'activité dans laquelle il est impliqué. La division du travail reflète les responsabilités des individus envers les autres membres de la communauté. Elle peut refléter les différents *rôles* que jouent les individus. Il faut noter que dans la littérature, les auteurs utilisent la notion de rôle pour ne décrire que la division du travail. Il est important de noter que, comme nous l'exposerons dans le chapitre 5, nous utiliserons pour notre conception une notion de rôle possédant un sens moins restrictif que celui relatif à la division du travail.

La Figure 7, représente le modèle présenté ci-dessus. Il faut noter que ce modèle apparaît toujours représenté sous une forme simplifiée pour en augmenter la lisibilité. Cependant, les auteurs précisent généralement de ne pas oublier que l'activité est un système complet et qu'il existe en réalité des relations entre chacun de ses éléments.

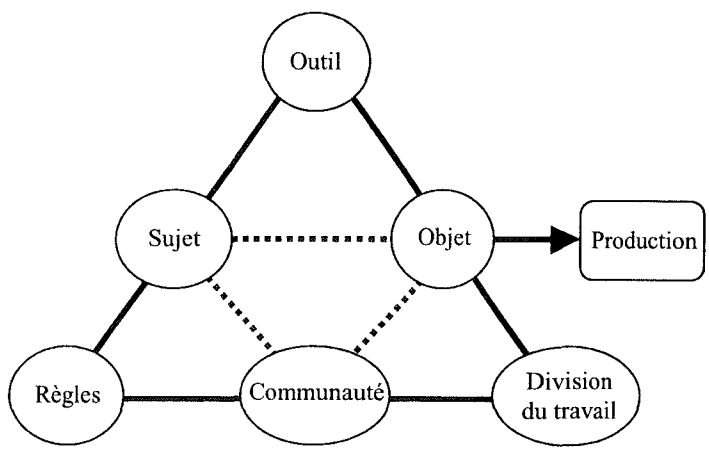

Figure 7. La structure basique d'une activité

3.4.3 Une constante évolution

Comme nous l'avons exposé, tous les éléments médiateurs du modèle de la structure basique d'une activité sont ouverts à de futures modifications. Ils constituent une situation qui influence l'activité. Réciproquement, ces éléments sont développés par l'activité dans un phénomène réflexif. De plus, les autres éléments du modèle que sont l'objet, le sujet et la communauté sont eux aussi susceptibles d'évoluer au cours de l'activité.

La relation qui existe entre le sujet et l'objet est une relation réciproque. En effet, le sujet transforme l'objet de l'activité, mais en même temps, les propriétés de l'objet transforment le sujet en augmentant son expérience. Ainsi, comme le souligne Kuutti [68], la Théorie de l'Activité rejette la première notion de modèle mental statique et définit la cognition comme un processus situé. Dans la même idée, Bellamy [5] rappelle que la nature bidirectionnelle de la médiation affecte les processus mentaux du développement de l'individu.

L'objet de l'activité lui-même peut être modifié et sa structure n'est pas immuable [68]. La modification de l'objet n'est pas un processus continu. Il passe toujours par des périodes de stabilité qui permettent ainsi à l'activité de maintenir une certaine direction. Cependant, l'objet peut s'avérer irréalisable. Il peut ne plus répondre à un besoin ou encore être en contradiction avec l'objet d'une autre activité à laquelle une partie des sujets participent. Nardi [95] précise que la modification de l'objet n'est pas triviale et qu'elle change fondamentalement la nature de l'activité qu'il motive, remettant en cause tous les éléments qui la composent. Cette décision pourra être prise suite à un processus de négociation, au cours de l'activité, et en fonction de sa situation (outils, règles, etc.).

Enfin, la communauté représente l'ensemble des sujets partageant le même objet d'activité. Cette communauté peut subir de nombreuses modifications, en particulier le départ ou l'arrivée de nouveaux sujets. Ces modifications peuvent entraîner des changements concernant les autres éléments de l'activité, comme par exemple la création de nouveaux rôles ou encore une redéfinition de la division du travail.

Ainsi, tous les éléments du modèle peuvent évoluer au cours de l'activité. Ces changements concernent aussi bien le sujet, l'objet et la communauté, que les éléments médiateurs (l'outil, les règles et la division du travail). Les conflits émergeants internes à l'activité en sont les principaux facteurs de transformation. Par exemple, un jeu de règles mis en place au sein d'une communauté peut entrer en contradiction avec une certaine division du travail, auquel cas des modifications doivent être effectuées. Les contradictions existantes entre des activités

différentes constituent aussi une force de développement. Nous sommes tous, chaque jour, impliqués dans plusieurs activités parallèles ou concurrentes. Il arrive souvent que des conflits surgissent entre elles, nous amenant à vouloir les modifier, les adapter.

Pour conclure, nous pensons qu'en vertu de sa simplicité, ce modèle peut favoriser le partage d'une compréhension commune entre l'informaticien ou le concepteur logiciel, le chercheur en sciences humaines, ou encore la communauté des utilisateurs. Il a d'ailleurs été fortement utilisé dans le domaine des Interactions Homme Machine depuis quelques années, domaine qui, comme nous le savons, possède en commun avec le TCAO la particularité de mêler à la fois les sciences humaines et celles de l'informatique. En particulier, ce modèle a été fortement utilisé par K. Kuutti.

3.5 Les apports de Kuutti

3.5.1 L'activité comme unité de base d'analyse du TCAO

En 1991, Kuutti [69] explique que le nouveau champ de recherche que constitue le TCAO souffre du problème de la sélection d'une unité basique structurelle et fonctionnelle pour analyser le travail. Plus encore, il identifie qu'il n'existe pas encore de définition adéquate et universelle de ce que signifie, ou de ce qu'on entend par « travail coopératif ».

Après avoir exposé la difficulté persistante à définir le TCAO, même si de nombreux chercheurs tels que Bannon et Schmidt, Lyytinen et Suchman se sont attelés à cette tâche, Kuutti propose lui-même une définition qui serait susceptible de synthétiser toutes les autres : le TCAO peut être défini comme *le travail de multiples sujets actifs qui partagent un objet commun et supporté par la technologie de l'information*. Il souligne que l'élément clé de cette définition est le concept de *sujets actifs* qui permet de différencier le TCAO des systèmes d'information traditionnels dans lesquels la prédétermination de séquences de travail par le système correspond au cas normal d'exécution. Dans cette définition, l'auteur utilise le concept de sujet actif tel qu'il est défini dans l'AT, c'est-à-dire un sujet qui est en accord avec l'objet de son activité et le comprend. Ainsi, dans le but d'isoler un contexte minimal pour l'étude du travail, il définit l'activité, telle qu'elle apparaît dans le modèle d'Engeström, comme unité basique d'analyse pour la recherche en TCAO.

3.5.2 L'expansivité

La seconde contribution de Kuutti qui nous semble très importante consiste en une typologie ou une classification des types basiques de support au travail, du point de vue de la technologie de l'information. Il résume cette classification dans un tableau (cf. Tableau 2) dont les colonnes représentent les parties principales qui constituent une activité (cf. *la structure basique d'une activité*). En particulier, la colonne nommée sujet 'pensant' est utilisée pour représenter l'activité interne du sujet. Ce point est très significatif car rappelons que dans l'AT, le contexte est à la fois interne aux personnes (impliquant des objets spécifiques et des buts) et externe (impliquant des artefacts, d'autres personnes et des configurations spécifiques). Les lignes du tableau représentent les différentes attitudes qu'une personne est susceptible d'adopter au cours de l'activité. Le Tableau 2 constitue la fusion de deux versions de la classification de Kuutti qui sont présentées dans [69] et [68]. On peut remarquer que dans la seconde version, Kuutti remplace les noms des lignes *Passif*, *Actif* et *Expansif* respectivement par *Support au niveau opération*, *Support au niveau action* et *Support au niveau activité*. De cette manière, l'auteur replace sa classification dans le cadre des niveaux de la structure interne de l'activité (cf. 3.2), la rendant plus proche de l'AT elle-même. Cependant, nous avons préféré conserver la première version, celle-ci permettant de mieux appréhender la notion d'expansivité qui se trouve au cœur de la conception de notre système de support au TCAO.

	Outil	**Règles**	**Division du travail**	**Sujet «pensant»**	**Objet**	**Communauté**
Passif	Automatisation de routine	Contrôle implicite et imposé	Fixée et imposée	Déclenche des actions prédéterminées	Données	Implicite et invisible
Actif	Support aux actions de transformation et de manipulation. Rendre les outils et procédures visibles	Pensées partagées et visibles	Coordination, Organisation visible du travail	Recherche l'information	Matériel partagé	Réseau visible Support à la communication
Expansif	Automatisation De nouvelles routines ou construction d'outil	Construction, négociation des règles	Réorganisation	Apprend, comprend	Construction de l'objet	Construction de la, ou d'une nouvelle communauté

Tableau 2. Une classification des types basiques de support au travail

Cette classification s'avère d'une grande utilité pour situer notre travail par rapport aux approches TCAO 'classiques'. La première ligne peut être considérée comme une représentation des systèmes d'information traditionnels adoptant une approche Tayloriste du travail. On peut par exemple y associer les premiers systèmes de *Workflow*. La seconde ligne correspond, selon la définition donnée par l'auteur, aux systèmes 'classiques' de support au TCAO, c'est-à-dire ceux mettant en jeu des sujets actifs partageant un objet commun. Enfin, la troisième ligne décrit une approche hors normes dans la recherche en TCAO et qui correspond à celle que nous avons adoptée pour nos travaux. *Cette approche définit le système ultime de TCAO comme celui qui permettrait la reconstruction du travail en créant les artefacts informatiques nécessaires, cette reconstruction étant réalisée par les travailleurs eux-mêmes* [69].

3.5.3 Une approche organisationnelle

La dernière contribution de Kuutti que nous présenterons utilise l'AT pour replacer les activités dans une analyse organisationnelle [70]. Dans les situations réelles, il existe toujours un réseau d'activités interconnectées. Kuutti rappelle que la participation à diverses activités interconnectées, possédant des motifs très différents, peut causer des tensions et des distorsions. Les connexions peuvent former un réseau hiérarchique et, puisqu'une activité délimite une partie des processus de travail dans une organisation, ce concept peut être utilisé pour analyser les organisations.

Kuutti définit l'organisation comme un réseau hiérarchique d'activités interconnectées. Il décrit alors un ensemble non exhaustif de types de connexions. Le principe de base est que chacun des éléments d'une activité (ceux apparaissant dans la structure basique d'une activité d'Engeström) peut constituer l'objet d'une ou de plusieurs autres activités dans l'organisation. Par exemple, un service peut avoir comme objet de produire des outils qui, une fois réalisés, médiatiseront une autre activité. On peut aussi considérer le cas d'un service de management qui a pour objet d'activité de nombreuses 'sous' activités, faisant ainsi naître une structure hiérarchique dans l'organisation. Enfin, l'auteur considère le schéma générique dans lequel une activité produit un résultat qui devient l'objet d'une autre, et ainsi de suite...

De cette manière, Kuutti expose comment modéliser et/ou analyser des organisations. Il évoque les problèmes qui peuvent être appréhendés grâce à une telle modélisation. Par exemple, et comme nous l'avons exposé précédemment, une certaine culture se développe au sein de chaque activité (au sein d'un département par exemple). Cette culture peut alors

entrer en conflit avec celle d'autres activités. Kuutti expose alors les différents points qui doivent être pris en compte par les systèmes de support aux organisations. Pour ce faire, il réutilise sa classification des types basiques de support au travail présentée ci avant, et insiste particulièrement sur la notion d'expansivité ou *« expansiveness »* (troisième niveau de la classification) qui, si elle est supportée par les systèmes, peut apporter une solution aux problèmes émergeants dans l'organisation supportée. Notre but étant aussi de supporter une organisation, cet apport pourra se révéler d'une grande utilité.

3.6 Coordination, coopération et co-construction

A l'instar de Kuutti, qui en 1991 s'interrogeait sur la définition universelle du travail coopératif, sept ans plus tard, Jacob Bardram expose le problème persistant de la recherche à comprendre la nature du TCAO [2].

Pour exposer son approche, Bardram se base sur la structure hiérarchique à trois nivaux de l'activité (cf. 3.2). Bardram [1] souligne que l'*anticipation* (ou une réflexion d'anticipation) guide l'activité dans ses trois niveaux : l'anticipation motive l'activité (anticipe la réalisation de l'objet), elle est le but de l'action et la base d'orientation des opérations (cf. 3.2.2). L'anticipation guide donc toute l'activité en créant une synthèse entre la perception de l'état environnemental (la situation) et l'expérience du sujet. Comme nous l'avons décrit précédemment, la situation ne cesse d'évoluer au cours de l'activité. Il en est de même pour l'expérience du sujet. C'est pourquoi, une anticipation n'est réellement valide qu'au moment de sa construction, en particulier lorsqu'il s'agit d'une anticipation à long terme comme un plan d'actions.

« the computer should be a tool mediating the anticipatory reflection of recurrent events in working life. Hence, such a planning tool should support situated planning – building, altering, sharing, executing, and monitoring plans within the cooperative work activities » [1].

En conséquence, Bardram redéfinit les systèmes de Workflow comme des systèmes de support à la planification située. Les systèmes de Workflow mettent classiquement en œuvre des planifications trop rigides au regard de l'AT. Les plans d'actions sont des artefacts cognitifs ou matériels qui supportent l'anticipation et qui médiatisent l'activité. Comme tout élément médiateur, le plan d'action doit donc être révisable in situ. Un intérêt particulier est porté aux pannes qui peuvent survenir dans un plan d'actions. Celles-ci ne doivent pas être considérées comme des exceptions dans l'activité, mais comme des parties naturelles et très

importantes de celle-ci. Elles forment la base de l'apprentissage et donc du développement et de l'amélioration des plans pour les actions futures. Bardram souligne d'ailleurs lui-même le côté réflexif de l'activité : le processus d'apprentissage crée et améliore le plan qui, à l'origine, guidait l'activité.

Sur les bases de ces investigations menées dans le cadre particulier du Workflow, les idées sont étendues au TCAO en général [2]. Dans le travail coopératif, les routines, la coopération et les conflits ne doivent pas être vus comme des catégories distinctes de travail, mais comme différents aspects du même effort. La notion de dynamique ne doit pas être négligée par le TCAO car le schéma de coopération dans le travail est souvent ré-instancié en réponse à une situation conflictuelle.

Bardram utilise une structure hiérarchique définie par Engeström [39] pour décrire la dynamique des transformations existant dans l'activité. Cette structure comporte trois niveaux : la *coordination*, la *coopération* et la *co-construction* de l'activité. Malgré les apparences, ces trois niveaux se démarquent quelque peu des trois niveaux de la classification de Kuutti :

- La coordination correspond au niveau le plus bas où les individus ne font que se concentrer sur la réalisation d'actions prédéterminées par un script d'exécution. Ce niveau correspond bien au niveau 'passif' ou 'opération' de la classification de Kuutti.
- La coopération, niveau intermédiaire, est celui où les sujets sont actifs, c'est-à-dire qu'ils partagent l'objet de leur activité. Chaque sujet insère ses propres actions dans le flot d'actions des autres. L'auteur ajoute qu'il est parfois même nécessaire d'influencer, par ses propres actions, les actions des autres, ceci dans le but de réaliser la tâche commune. A ce niveau, le partage de l'objet permet aux sujets de se mettre en question les uns par rapport aux autres et d'apporter des corrections à leurs propres actions, ou celles des autres, pour rester en accord avec l'objet de l'activité collective. Même si ce niveau semble correspondre au niveau 'actif' ou 'action' de la classification de Kuutti, on peut noter une différence majeure. En effet, Bardram précise que la recherche d'artefacts appropriés pouvant être utilisés en tant que médiateurs de l'activité est réalisée à ce niveau. Or, pour Kuutti, ce point apparaît au niveau supérieur et se traduit en terme d'expansivité.
- La co-construction représente les interactions dans lesquelles les sujets re-conceptualisent leur organisation et leurs interactions en relation à l'objet partagé. A ce niveau, l'objet de l'activité n'est pas stable ou n'existe peut-être même pas. Il doit être construit

collectivement par les sujets participants. Ceci correspond à l'expansivité décrite par Kuutti mais principalement focalisée sur l'objet de l'activité lui-même.

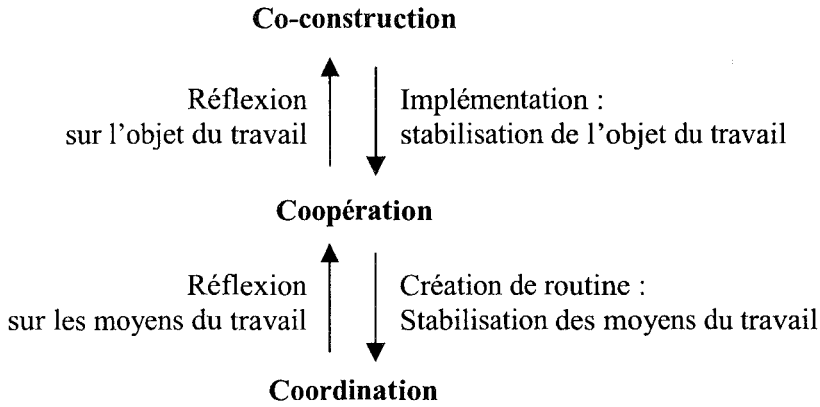

Figure 8. Les dynamiques du travail coopératif

Bardram expose alors les différents types de transitions, représentés Figure 8, qui permettent de passer d'un niveau à l'autre :

- Le passage du niveau coordination au niveau coopération est nommé *réflexion sur les moyens du travail*. Les moyens en question sont les artefacts qui médiatisent l'activité c'est-à-dire les outils, les règles et la division du travail, même si l'auteur ne les cite pas tous dans sa définition. Au cours de cette phase, les éléments médiateurs sont redéfinis coopérativement et de manière à rester en accord avec l'objet de l'activité. La transition descendante correspondante est appelée *création de routines* et rétablit un travail coordonné dont les moyens sont stables. Il est essentiel que, dans cette phase, tous les sujets soient au fait des modifications qui ont été apportées au niveau supérieur (la nouvelle division du travail par exemple).
- La *réflexion sur l'objet du travail* représente une transition amenant du niveau coopération au niveau co-construction. Un tel mouvement est nécessaire lorsque l'objet du travail, sur lequel reposent les niveaux coordination et coopération, se révèle instable (des motivations conflictuelles par exemple). L'*implémentation*, transition inverse, correspond à une stabilisation de l'objet de l'activité et implique la communication de ce nouvel objet aux sujets actifs de l'activité.

Ayant utilisé cette structure et ces mécanismes pour définir un outil de TCAO (le *Patient Scheduler*) destiné à supporter la coordination des personnels d'un hôpital et à analyser cette coordination, Bardram conclut [2] en proposant deux principales propriétés que doivent posséder les systèmes de support au TCAO : *(1) supporter les activités de travail collaboratif dans les trois niveaux, ainsi que les transitions entre ceux-ci, et (2) intégrer ce support à la coopération dynamique dans le support de l'objet du travail lui-même.* En d'autres termes, les coordination, coopération et co-construction du travail doivent être supportées par le système, et ces fonctionnalités doivent être, du point de vue du sujet (et pas seulement du développeur), intimement liées à l'activité supportée.

3.7 La théorie instrumentale

Une approche quelque peu parallèle à l'AT, et pouvant même être considérée comme une extension de celle-ci est récemment apparue dans la communauté française. Il s'agit de l'approche instrumentale principalement introduite par P. Rabardel et P. Béguin [4]. Cette nouvelle approche propose principalement deux modèles.

Tout d'abord, le modèle SAI (Situations d'Activités Instrumentées) dans lequel l'outil, ou plus précisément la notion d'artefact médiateur de l'AT, est redéfini en terme d'instrument. La différence majeure entre l'artefact et l'instrument est que ce dernier est bipolaire, tenant à la fois du sujet et de l'objet. L'instrument contient de manière indissociable les entités *artefact* (permettant d'agir sur l'objet), et *mode d'usage* (sous la forme de schèmes d'utilisations). L'appropriation de l'artefact par le sujet au cours de l'activité est alors considérée comme une genèse instrumentale, notion qui peut être rapprochée de la création d'organes fonctionnels définie par l'AT.

Le modèle SACI (Situation d'Activités Collectives Instrumentées) est une extension du modèle SAI pour le replacer dans un contexte de groupe et a été utilisé dans l'analyse de collecticiels [14]. Cette extension peut rappeler celle apportée au modèle de l'activité individuelle médiatisée d'Engeström pour le transformer en celui que nous utilisons dans la suite de nos travaux, c'est-à-dire la structure basique d'une activité. Cette similitude n'est toutefois pas très profonde car, selon les auteurs, le modèle d'Engeström, dans son approche systémique, contribue à une dissolution du sujet psychologique à laquelle l'approche instrumentale tente justement de remédier.

Même si les concepts et mécanismes exposés dans la théorie instrumentale nous semblent très intéressants, nous avons préféré utiliser le modèle d'Engeström pour modéliser DARE. En effet, l'approche instrumentale expose clairement comment un sujet intègre des artefacts dans son activité pour en faire des instruments, comment il crée et/ou modifie à la fois ses propriétés physiques et ses schèmes d'utilisation. Ces notions se révèlent d'un grand intérêt dans le cadre de DARE puisqu'un de ses buts est de permettre aux sujets d'intégrer dans l'environnement de support au TCAO, eux-mêmes et au cours de l'activité, les artefacts dont ils ont besoin en spécifiant, en fonction de la situation, la manière dont ils vont les utiliser (l'explicitation, in situ, de schèmes d'utilisation). L'approche instrumentale est particulièrement appropriée à l'analyse des systèmes de TCAO. Cependant, malgré les critiques justifiées faites au modèle d'Engeström, celui-ci, de par sa perspective systémique, nous paraît plus simplement utilisable pour le partage d'une représentation de l'activité coopérative avec l'utilisateur final. Nous envisageons néanmoins d'explorer à l'avenir les nouvelles pistes de travail introduites par la théorie instrumentale.

De plus amples informations sur la théorie instrumentale, ainsi qu'un bon exemple d'utilisation pour l'analyse de collecticiels peuvent être trouvées dans [14].

3.8 Les enjeux du projet DARE

Nous venons de présenter la Théorie de l'Activité et ses apports identifiés au domaine du TCAO. A partir de cette étude, il nous est possible de redéfinir notre propre vision de ce que doit être un environnement de TCAO. Notre but est d'identifier une série d'enjeux à atteindre pour que notre réalisation soit en adéquation avec les fondements de l'activité humaine.

3.8.1 Supporter l'activité

3.8.1.1 Notre vision du collecticiel

« we define CSCW as work by multiple active subjects sharing a common object and supported by information technology » [69]

Cette citation de Kuutti résume assez bien notre vision du collecticiel. Il s'agit de fournir un support à l'activité telle qu'elle est définie dans l'AT. Cependant, il faut d'ores et déjà bien différencier la notion de support à l'activité et l'activité elle-même. Le support à l'activité qu'est le collecticiel ne peut contenir qu'une représentation partielle de l'activité réelle. La

réalité inclut les sujets, leurs interactions non médiatisées informatiquement, leurs processus mentaux internes, etc. Nous pensons qu'il serait vain de tenter de réaliser une application dont le but est d'automatiser l'activité tout entière. Au contraire, le collecticiel doit fournir un support informatique aux besoins qui sont exprimés par les sujets et qui émergent au cours de l'activité réelle.

Au regard de l'AT, les besoins des sujets peuvent principalement être formulés en une explicitation d'éléments médiateurs de l'activité réelle. Toute partie rendue explicite peut potentiellement être supportée par le collecticiel. Par exemple, nous avons vu que la création d'une communauté est réalisée par le partage d'un objet d'activité. De par sa nature, le collecticiel peut aider à partager cet objet, si celui-ci est explicité. Dans le même esprit, un ensemble d'outils pour supporter l'activité peut être défini. Le collecticiel joue dans ce cas de rôle d'un environnement global dans lequel les sujets peuvent trouver des outils partagés leur permettant d'atteindre leurs buts. Plus encore, une partie des règles existantes dans cette nouvelle communauté peut être explicitée, ce qui aura pour effet d'automatiser certaines contraintes dans l'environnement de TCAO. On pourra par exemple limiter l'accès aux outils précédemment définis pour des sujets particuliers. Enfin, si une division du travail est arrêtée, le collecticiel peut une fois encore jouer un rôle dans son support quant à la réalisation de l'activité. Nous verrons par la suite plus en détail comment il est possible de supporter ces différentes dimensions. Ce qui nous importe ici, est de montrer que l'activité existe au-delà du collecticiel. Dans notre approche, celui-ci n'introduit des contraintes, aussi bien en termes d'outils, de règles ou de division du travail, que si elles ont été explicitement spécifiées par les sujets au cours de leur activité coopérative. De notre point de vue, ces contraintes correspondent à des besoins des sujets. C'est bien le sujet qui utilise le collecticiel pour réaliser son activité et non pas l'inverse. Bien entendu, cette vision peut paraître utopique dans le sens où l'utilisation du collecticiel impose inévitablement certaines contraintes ou limitations. Toutefois, notre but est de limiter au maximum ces effets indésirables de manière à aider l'activité plutôt que la gêner.

3.8.1.2 Un collecticiel générique et malléable

Une première approche dans le but de ne pas limiter les sujets dans l'utilisation du collecticiel et d'offrir un support à la plus large gamme d'activités possibles. A l'image de Dourish, qui pour la réussite de Prospero a opté pour l'approche la plus générique possible, il nous faut fournir un collecticiel qui permette de supporter tout type d'activité. C'est pourquoi l'AT, en

particulier la structure basique de l'activité définie par Engeström, nous est d'un apport inestimable : elle définit l'activité de manière générique, identifiant clairement un système d'éléments qui y participent. Cependant, l'apport d'AT est plus large que cela.

Le but principal du collecticiel est d'offrir un support à l'activité coopérative et de ce fait, il y participe. Son introduction correspond à l'introduction d'un nouvel outil dans l'activité. Selon l'AT, tout élément médiateur influence le cours de l'activité et il est a priori impossible de prédire son impact sur une activité particulière. C'est pourquoi *le collecticiel doit être considéré comme un élément médiateur à part entière.* En d'autres termes, s'il influence l'activité coopérative, il doit pouvoir être modifié par celle-ci. De plus, et d'une manière générale, l'AT montre que les éléments qui constituent l'activité ne cessent d'évoluer. Ceci implique que ceux qui ont été introduits par les sujets dans le collecticiel doivent pouvoir être re-spécifiés. Ces éléments sont aussi bien l'objet de l'activité, les outils qui sont utilisés, les règles exprimées, ou encore la division du travail mise en place. Le système de TCAO doit donc être assez malléable pour pouvoir être pratiquement totalement redéfini au cours de sa propre exécution. Cette malléabilité sera étudiée en détail dans le chapitre 4. Les seuls invariants sont la structure basique de l'activité, c'est-à-dire son modèle le plus générique, ainsi que les mécanismes de base qui la font vivre.

Rappelons que la redéfinition des éléments médiateurs correspond elle-même à une partie de l'activité coopérative. L'expansivité de Kuutti, et les dimensions coopératives et co-constructives exposées par Bardram nous montrent bien que le processus d'évolution est un processus coopératif, réalisé en fonction de la situation. Les règles et la division du travail existantes peuvent donc aussi s'appliquer à la manière dont la communauté fait évoluer sa propre activité. Dans la gestion d'un projet par exemple, le chef du projet aura certainement plus d'influence sur les réorientations potentielles que les autres participants. Ceci découle directement des règles et de la division du travail en pratique. Ainsi, la facette du collecticiel qui permet de re-spécifier les éléments qu'il met en œuvre doit être utilisée in situ et de manière coopérative. De ce point de vue, le collecticiel est utilisé pour définir sa propre utilisation. On voit ici à nouveau apparaître les mécanismes réflexifs inhérents à l'activité humaine.

3.8.1.3 Cristalliser l'expérience des sujets par rapport à l'activité

Nous avons vu qu'une partie des éléments qui constituent l'activité existe dans le collecticiel. Ces éléments y ont été introduits par des sujets exprimant leur besoin d'un support informatique envers ceux-ci. De plus, l'activité étant en constante évolution, les sujets ne cessent de les re-spécifier. Nous définissons un *type d'activité* comme un ensemble de spécifications d'éléments qui composent l'activité. Ces éléments sont un objet d'activité, des outils, certaines règles et une division du travail qui découlent de l'objet. Cet ensemble est construit en fonction de l'expérience des sujets. Il est donc possible de considérer un type d'activité comme un héritage culturel qui évolue au cours de chaque nouvelle activité de ce type. Un type d'activité n'existe que momentanément, par exemple entre deux activités : l'une s'étant terminée dans un certain état qui sert de base au démarrage d'une autre activité. L'expansivité de Kuutti peut être vue comme l'évolution d'un type d'activité vers un autre. Plutôt que de parler de type d'activité, on peut alors parler de *patron d'activités*.

Il est vain de croire que les sujets voulant utiliser le collecticiel pour une activité particulière vont totalement redéfinir, à chaque fois, tous les éléments qui doivent participer à leur activité. Il est plus probable que l'activité démarre selon un patron qui sera spécialisé in situ. De ce point de vue, le collecticiel est utile dans le sens où il permet de *cristalliser l'expérience* d'une communauté envers une activité dans un *patron d'activité réutilisable et adaptable*. Ceci permet aux sujets de *capitaliser* une expérience en y incorporant la leur.

3.8.2 Synthèse pour notre conception

Nous avons exposé les différents objectifs à atteindre pour poser les bases à la conception de DARE, notre collecticiel. Ces objectifs ont été directement inspirés par l'AT et nous espérons ainsi offrir une réponse adéquate aux problèmes actuels du TCAO.

Cette étude a permis de construire, par rapport aux problèmes qui nous intéressent, notre définition d'un collecticiel. Cette définition peut être résumée par les quatre points présentés dans le Tableau 3.

- Le collecticiel n'est pas l'activité, mais est un support à l'activité.
- Le collecticiel est un élément médiateur à part entière de l'activité.
- Le collecticiel supporte sa propre activité coopérative de redéfinition.
- Le collecticiel met en œuvre des patrons d'activité réutilisables et adaptables in situ.

Tableau 3. Notre définition d'un collecticiel, du point de vue de la Théorie de l'Activité

De manière à réaliser un tel environnement de support au TCAO, nous définissons cinq enjeux spécifiques qui doivent être atteints. Ces enjeux apparaissent dans le Tableau 4. Ils indiquent les directions à suivre dans notre réalisation d'un environnement de TCAO fondé sur la Théorie de l'Activité : l'environnement DARE (Activités Distribuées dans un Environnement Réflexif).

1) Offrir un support à l'activité selon les besoins exprimés par les sujets.
2) Supporter la redéfinition du support à l'activité au cours de l'activité.
3) Cristalliser l'expérience des sujets sous forme de patrons d'activités.
4) Permettre la réutilisation de patrons pour démarrer une nouvelle activité.
5) L'évolution d'une activité doit être un processus coopératif réalisé in situ.

Tableau 4. Les enjeux à atteindre dans la conception d'un collecticiel

Nous pensons que pour être utile et accepté par la communauté des utilisateurs, l'environnement de TCAO doit répondre à l'ensemble des points précités. Il nous faut donc définir un modèle de base ainsi qu'une architecture logicielle d'implémentation fournissant les concepts et mécanismes fondamentaux nécessaires au développement et à l'évolution d'un tel environnement. Pour ce faire, nous devons étudier les moyens identifiés dans le domaine de l'ingénierie logicielle concernant les systèmes informatiques évolutifs. De plus, d'après les enjeux que nous nous sommes fixés, ce sont les utilisateurs qui font évoluer leur propre système de TCAO. Cette propriété se traduit en terme de malléabilité.

Chapitre 4

Un système malléable

Le terme 'malléable' a déjà été introduit dans le chapitre 1. En effet, dans la définition d'ODESCA, une certaine malléabilité avait été identifiée comme propriété nécessaire des environnements de Travail Coopératif Assisté par Ordinateur (TCAO). Néanmoins, notre étude d'ODESCA avait permis d'identifier que la malléabilité proposée s'avérait assez faible par rapport aux besoins des utilisateurs.

L'étude de l'impact des sciences humaines sur le TCAO dans le chapitre 2 et de la Théorie de l'Activité dans le chapitre 3 nous ont montré que la malléabilité est une propriété fondamentale qui doit être prise en compte dans la création de systèmes de TCAO. En particulier, les dimensions expansive et co-constructive de l'activité nous démontrent que tout système de support à l'activité coopérative doit être *très* malléable.

Après avoir défini la malléabilité et rappelé son importance dans notre domaine de recherche, nous étudions les approches composants et les mécanismes associés qui permettent de la réaliser. Nous présentons ensuite les architectures cadres en insistant sur leur utilité dans la création de systèmes de TCAO malléables. Enfin, nous proposons notre propre approche qui tente de lier le cadre conceptuel défini par la Théorie de l'Activité et une architecture cadre destinée à supporter notre réalisation d'un environnement de TCAO : DARE.

4.1 La malléabilité

4.1.1 Définition

"End-user tailoring is the term I use [...] to describe the activity of adapting generic computer applications to local work practices and user needs" [90]

"The terms 'tailoring', 'end-user tailoring' and 'end-user redesigning' [...] will be used to denote the same thing" [87]

La définition de 'tailoring' proposée par Anders Morch correspond tout à fait à nos attentes pour décrire les propriétés devant être supportées par DARE. A notre connaissance, il n'existe pas de mot français équivalent à 'tailorability'. Il convient alors d'employer des termes comme 'adaptabilité' ou 'flexibilité'. Malheureusement, un seul de ces termes ne suffit pas à décrire la signification exacte de 'tailorability' qui possède aujourd'hui un sens bien défini dans notre domaine de recherche. En particulier, 'tailorability' introduit les notions d'adaptabilité et de flexibilité offertes aux utilisateurs finaux[1] d'une application informatique. C'est pourquoi, nous choisissons d'utiliser le terme *malléabilité*, en tant que terme équivalent à celui défini par Morch. Une application possédant une telle propriété sera appelée une application *malléable.*

"- Tailoring-as-design. Tailoring is continued development of an application by making persistent modifications to the application, as opposed to the products created with it.
- Tailoring-as-use. Tailoring is initiated in an application being inefficient or difficult to use for a specific task at hand" [88]

La malléabilité introduit des caractéristiques particulières. Elle ne correspond pas exactement à la possibilité de développer l'application, ni d'utiliser l'application, mais sous-entend un peu des deux. Une application malléable est à la fois utilisable et modifiable par ses propres utilisateurs. L'activité de redéfinition correspond elle-même à une des facettes de son utilisation. Il serait difficile d'ignorer un lien direct avec DARE dont les fondements, basés sur la Théorie de l'Activité (l'AT), décrivent des mécanismes permettant aux sujets/utilisateurs de redéfinir leur propre collecticiel au cours de son utilisation (cf. chapitre 3).

[1] Les utilisateurs finaux sont ceux qui utilisent l'application comme un support à leur travail. A l'opposé, les informaticiens utilisent l'application dans le cadre de son développement. L'utilisateur final dans le cadre de nos travaux correspond au sujet impliqué dans l'activité coopérative.

La malléabilité n'implique pas obligatoirement la possibilité de redéfinir l'application malléable au cours de sa propre exécution. Certaines réalisations peuvent nécessiter de redémarrer l'application pour que les modifications soient prises en compte. Dans certains cas, des parties du système peuvent même demander à être recompilées. Cependant, il a souvent été identifié dans le domaine du TCAO que la malléabilité 'à chaud' est favorable au support des activités coopératives. De cette manière, les sujets peuvent adapter leur collecticiel sans devoir interrompre leur activité.

La malléabilité, grâce à des auteurs comme Malone [79], est aujourd'hui considérée comme une propriété indispensable qui doit être prise en compte dans tout processus de conception et développement de collecticiel.

4.1.2 La malléabilité et les sciences humaines

Dans le cadre de DARE, nous sommes arrivés à la malléabilité suite à nos investigations assez approfondies dans les sciences humaines et en particulier dans l'AT. Parallèlement, de nombreuses autres approches éclairées par des résultats empiriques se sont orientées vers la malléabilité pour atteindre leurs objectifs. Elles nous semblent certainement aussi quelque peu influencées par les travaux de Suchman.

"Flexibility and tailorability are critical factors for the success of interactive systems. For CSCW systems these properties are even more important since several people have to be satisfied at once." [66]

Cette citation est caractéristique de l'approche classique actuelle de la malléabilité dans la recherche en TCAO. Une remarque nous semble intéressante : la dérive générale du TCAO vers la malléabilité propose souvent des solutions faiblement ancrées dans les sciences humaines. Ces dernières sont utilisées pour extraire des résultats dans des études empiriques, mais elles sont plus un point de départ validant une certaine intuition qu'un réel fondement pour la conception du système malléable. Si l'influence des sciences humaines sur ces travaux est indéniable, il est difficile d'identifier un lien profond entre leurs fondements théoriques et la construction de l'application malléable. Ces travaux focalisent plus spécifiquement sur les solutions techniques permettant de supporter la malléabilité, plutôt que sur les fondements de cette malléabilité. En conséquence, les questions en suspens portent souvent sur la place de la malléabilité dans les collecticiels. Les différentes contributions semblent s'accorder sur le fait

qu'un collecticiel ne doit pas être totalement modifiable. La question est : où placer les points de modifications[1] ?

Ces contributions n'en sont pas moins intéressantes et offrent un contexte dans lequel il est possible de replacer nos propres travaux qui adoptent une démarche quelque peu différente : nous tentons de marier la malléabilité avec les notions et mécanismes identifiés dans l'AT.

4.1.3 La malléabilité au cœur de différents domaines

Le problème de la malléabilité rejoint partiellement trois larges domaines de recherche : la réutilisation logicielle, la conception participative et la programmation par les utilisateurs [90]. Pour notre part, nous nous intéresserons principalement à la réutilisation logicielle et à la programmation par les utilisateurs.

La conception participative (CP), même si elle se révèle d'une grande aide pour la conception logicielle, ne rejoint par directement nos objectifs. La CP traite de l'implication des utilisateurs dans le processus de développement de leurs logiciels en les mettant en contact avec les concepteurs ou développeurs. Une question générale de la CP est comment supporter au mieux la coopération entre les informaticiens, les utilisateurs et éventuellement les chercheurs en sciences humaines. Un bon exemple se trouve dans les travaux de Bodkers. Dans le cadre du projet Eurocode [11], l'auteur présente une boîte à outils conceptuelle destinée à supporter la conception d'applications de TCAO. Cette boîte à outils favorise la communication entre les concepteurs et les utilisateurs en utilisant une approche de conception expérimentale. Elle permet aux utilisateurs finaux de spécifier leurs besoins au travers de scénarios, de listes de contrôle et de prototypes [10]. Dans cette configuration, l'utilisateur et le concepteur sont mis en contact pour créer le collecticiel. Notre recherche se situe plus dans une approche permettant à l'utilisateur ou sujet de modeler lui-même son collecticiel au cours de son utilisation.

A l'inverse, nous nous intéressons fortement à la réutilisation logicielle qui permet de développer des applications à partir de substrats informatiques, ainsi qu'à la programmation par les utilisateurs qui doit leur permettre d'adapter leur application. *Dans DARE, le challenge à accomplir est d'offrir ces fonctionnalités sans que l'utilisateur ait l'impression de programmer.*

[1] Une fois que les points de modification ont été identifiés, une autre question est : comment donner les moyens d'interactions aux utilisateurs finaux pour qu'ils puissent accéder à la malléabilité du système ?

4.1.4 La réutilisation logicielle

Comme le souligne Morch [90], le problème de la réutilisation n'est pas nouveau dans le domaine de l'ingénierie logicielle. Il s'agit de définir les éléments informatiques qui permettent cette réutilisation. Morch identifie quatre moyens de réutiliser du logiciel. Le premier consiste en la réutilisation de librairies telles que nous les connaissons dans le langage C. Ce type de réutilisation nous semble aujourd'hui quelque peu désuet, en particulier depuis l'avènement du paradigme Objet. Ceci nous amène directement au deuxième moyen de réutilisation : l'héritage de classes existant dans les langages Objets tels que Smalltalk et Java. Sans trop prendre de risques, nous pouvons affirmer que le mécanisme d'héritage est aujourd'hui celui le plus utilisé dans la communauté informatique pour la réutilisation logicielle. Le troisième moyen de réutilisation considère l'objet (au sens de l'ingénierie logicielle) comme un élément qui s'exécute et interagit avec d'autres objets. Les objets ne sont pas classifiés en une hiérarchie mais vivent dans l'exécution du système. Ils peuvent survivre à une exécution en étant sauvegardés dans une base de donnée, et être réutilisés plus tard pour la construction d'un nouveau système. Morch précise qu'il n'existe pas aujourd'hui de mécanismes bien établis mettant en œuvre une telle réutilisation. Le quatrième et dernier moyen est celui qui utilise un générateur d'application. Ce générateur est un *méta-système* qui définit un langage de plus haut niveau (plus proche du domaine d'utilisation des applications générées) qu'un langage général de programmation. Les applications sont générées lorsque l'utilisateur remplit certains blancs. Ce qui est généralement réutilisé d'une application à l'autre est le générateur lui-même. Nous reviendrons par la suite sur la notion de méta-système car DARE est lui-même un méta-système qui, de plus, permet de réutiliser et spécialiser du code sous forme de classes.

4.1.5 La programmation par les utilisateurs

"End-user programming is the activity of writing (usually small, interpreted) programs in an end user programming language" [90]

La programmation par les utilisateurs est elle aussi identifiée comme un des domaines investis par la malléabilité. Une application malléable doit fournir aux utilisateurs un langage leur permettant d'expliciter leurs nouveaux besoins. La définition ci-dessus, tirée de la thèse de Morch, précède une large description des différentes recherches ayant été réalisées dans ce domaine. De ce fait, nous ne pensons pas qu'il soit nécessaire de reprendre de manière exhaustive toutes ces contributions. Nous aimerions plutôt souligner une remarque de

l'auteur : il existe toujours un lien difficile à établir entre l'expertise de l'utilisateur (envers la réalisation de sa tâche) et le langage proposé (qui doit suivre les règles d'une grammaire formelle) pour spécifier ses besoins.

4.2 La malléabilité par les composants

Un des problèmes liés à la malléabilité est de définir un élément de base permettant de la supporter. Cet élément doit être à la fois réutilisable, évolutif et compréhensible par les utilisateurs sensés le manipuler. De par son ampleur prise dans notre domaine, l'approche composant est une solution qui semble faire l'unanimité. Un composant est définit comme :

"[...] a unit of composition with contractually specified interfaces and explicit context dependencies only. Components can be deployed independently and are subject to composition by third parties." ([128], p.130)

La *programmation orientée composants* a sans doute été largement motivée par le succès rencontré par les composants standardisés couramment utilisés dans les domaines d'ingénierie plus classiques telle l'électronique (résistances, transistors, etc.) ou encore la mécanique (vis, boulons, etc.). Depuis longtemps, ces disciplines utilisent des composants plus ou moins simples pour construire des systèmes plus complexes. De cette notion découle la définition d'un *composant informatique*. Un système construit à partir de composants possède une *architecture composants*. Si le niveau d'abstraction envisagé pour décrire un tel système est le composant, sa description est réalisée grâce à un *langage de composition* [122]. Lorsque ce langage correspond à un langage de programmation orienté Objet classique et général tel Java, les composants sont des instances de classes. La classe est alors utilisée comme un générateur permettant d'instancier un nombre quelconque d'objets composants.

4.2.1 Des standards

Les espoirs de l'ingénierie logicielle envers les composants sont la facilité de réutilisation (spécialement par des tiers), la réduction du temps pris par le processus de développement, la réduction des coûts de développement et, pour finir, l'augmentation de la qualité des produits (des composants standards déjà testés menant à une réduction des erreurs possibles). Le succès des composants a déjà été démontré en informatique dans le domaine de la conception d'interfaces utilisateur avec des logiciels tels que Microsoft Visual Basic ou Jbuilder .

Forte de ces espoirs et réussites, l'ingénierie logicielle fournit encore aujourd'hui un effort considérable pour standardiser les composants. On peut par exemple citer certains standards de fait comme le Distributed Component Object Model (DCOM) de Microsoft [20], ou encore les JavaBeans [57], modèle de composant pour Java de Sun. Enfin, le modèle de composant fournit par l'OMG [100] dans le cadre de la Common Object Request Broker Architecture (CORBA)[48] promet un bel avenir aux approches composants, dans le domaine de la conception logicielle en général, et dans le TCAO en particulier.

4.2.2 La démarche générale du TCAO

Dans le cadre du TCAO, certaines propriétés des architectures composants ont été identifiés pour répondre aux problèmes du domaine. Pour qualifier l'approche générale des composants dans le TCAO, nous pouvons porter un bref regard sur celle mise en œuvre dans le projet EVOLVE [125]. Les auteurs précisent que, même si les applications sont aujourd'hui généralement réalisées à partir de composants, cette structure est souvent perdue dans l'application délivrée qui apparaît comme monolithique. Le but, dans la malléabilité par les composants, est de laisser transparaître la structure composants après le développement initial. Cette démarche offre un moyen simple d'évolution de l'application par la manipulations de sa structure au niveau d'abstraction des composants. La Figure 9 ci-dessous présente deux approches pour la construction d'applications à partir de composants. La partie gauche correspond à la construction d'une application monolithique. La partie droite décrit une application malléable. Notre approche suit la démarche exposée dans la partie droite.

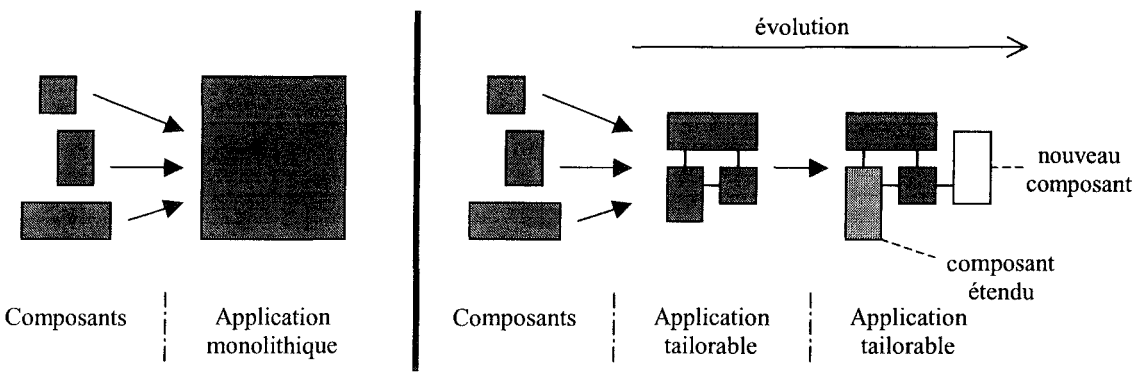

Figure 9. Application monolithique et application malléable construites à partir de composants (adaptées de [124])

4.2.3 La malléabilité hiérarchique

"The system architecture should allow tailoring activities at different (and appropriate) levels of abstraction and complexity" [123]

Une partie des concepteurs de POLITeam [65] ont récemment contribué à identifier certaines propriétés supportées par l'approche composant et qu'il est souhaitable de faire apparaître dans les architectures pour le TCAO. L'une de ces propriétés correspond à une approche hiérarchique de la malléabilité. L'argument principal est que l'activité de modification du système s'adresse aussi bien à de simples utilisateurs qu'à des administrateurs de systèmes ou encore à des tiers consultants. Ces différents types d'utilisateurs possèdent généralement différents points de vues sur l'application. Chaque point de vue est engendré par des compétences différentes.

Une découpe hiérarchique n'est envisageable que si l'architecture composant est elle-même bâtie sur un modèle hiérarchique. En d'autres termes, les composants d'une application sont eux-mêmes composés de composants à différents niveaux d'abstraction. Prenons l'exemple simple d'un composant applicatif tel une calculette.

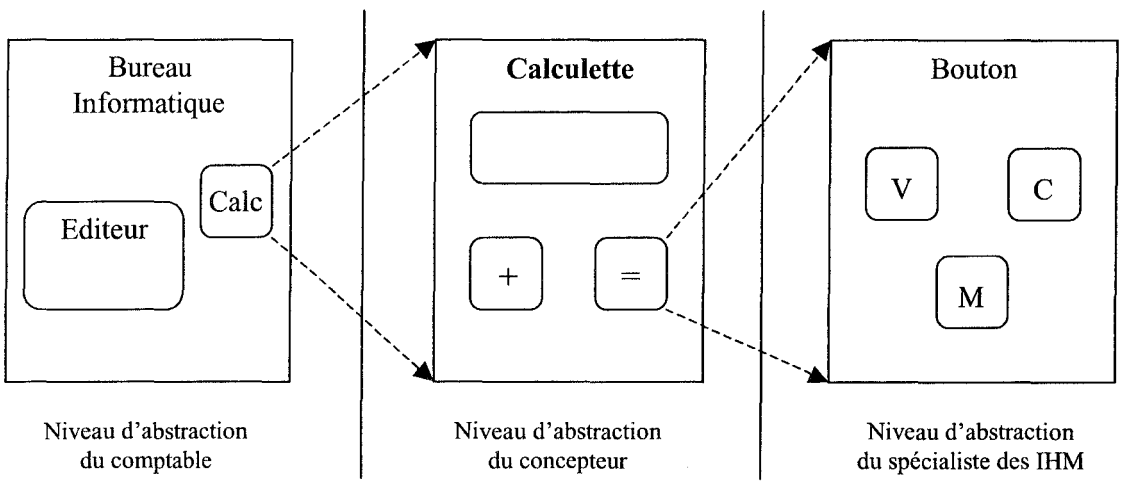

Figure 10. Une architecture hiérarchique de composants.
Différents points de vues selon l'expertise des sujets.

La Figure 10 montre la calculette comme un composant d'assez haut niveau pouvant participer à une application de style bureau informatique d'un niveau encore plus haut. Pour un comptable, la calculette est le niveau d'abstraction qui l'intéresse et, si le bureau informatique est malléable, celui-ci peut envisager de changer sa calculette pour une autre. La

calculette elle-même peut être constituée d'un ensemble de composants graphiques (widgets) comme des boutons. Pour le concepteur de la calculette, les boutons correspondent au niveau d'abstraction des composants qu'il peut décider d'enlever, de remplacer ou encore de déplacer. Enfin, le spécialiste des interfaces homme-machine considère le bouton lui-même comme un ensemble de composants implémentant MVC (Modèle Vue Contrôleur). Ce spécialiste peut décider de modifier ou remplacer un des composants du bouton si la malléabilité est encore accessible à ce niveau d'abstraction. Nous voyons bien ici comment différents acteurs possédants différents niveaux d'expertise peuvent envisager le système selon la hiérarchie de composants et la malléabilité présente. Chacun possède des compétences qui le place en spécialiste par rapport à un certain niveau d'abstraction : le comptable est par exemple certainement le plus compétent pour décider de quelle calculatrice il a besoin pour réaliser son travail.

Cet exemple simple démontre en quelle mesure une architecture composants peut introduire une malléabilité hiérarchique avec des niveaux d'abstraction intéressants pour des acteurs impliqués dans une même activité, mais possédant des compétences différentes. Un autre atout de la malléabilité hiérarchique est de permettre à un sujet de descendre pas à pas dans les niveaux de l'application. La maîtrise d'un certain niveau d'abstraction permet au sujet de comprendre plus aisément le niveau qui se trouve juste en dessous. Un niveau d'abstraction plus bas permet de modifier plus finement l'application. Ainsi, on peut imaginer qu'avec l'accroissement de son expérience, un sujet finira par maîtriser tous les niveaux de la malléabilité d'une application.

4.2.4 Le paramétrage, l'intégration et l'extension

Transversalement aux différents niveaux hiérarchiques qui peuvent apparaître dans une architecture composants, nous pouvons distinguer trois formes de malléabilité. Ces trois formes, à l'origine identifiés par Morch [88] sont : le paramétrage, l'intégration et l'extension de composants.

4.2.4.1 Le paramétrage

Le paramétrage est une forme de malléabilité aujourd'hui bien répandue et présente dans la plupart des applications informatiques, qu'elles soient coopératives ou non. Il s'agit de sélectionner une ou plusieurs valeurs d'un ensemble prédéfini qui permettent de spécialiser

l'application. Un exemple simple de paramétrage est la modification des couleurs d'une application. Dans Netscape Navigator, il peut s'agir d'un affichage des boutons d'une barre d'outil avec ou sans texte (cf. Figure 11). Pour un outil coopératif tel un outil de vote, il peut s'agir d'un type de coopération prédéfini pour l'activité des participants. A un tout autre niveau d'abstraction, il peut aussi s'agir de choisir une certaine police de caractères pour un bouton sous forme de JavaBeans [57] dans la BeanBox de Sun.

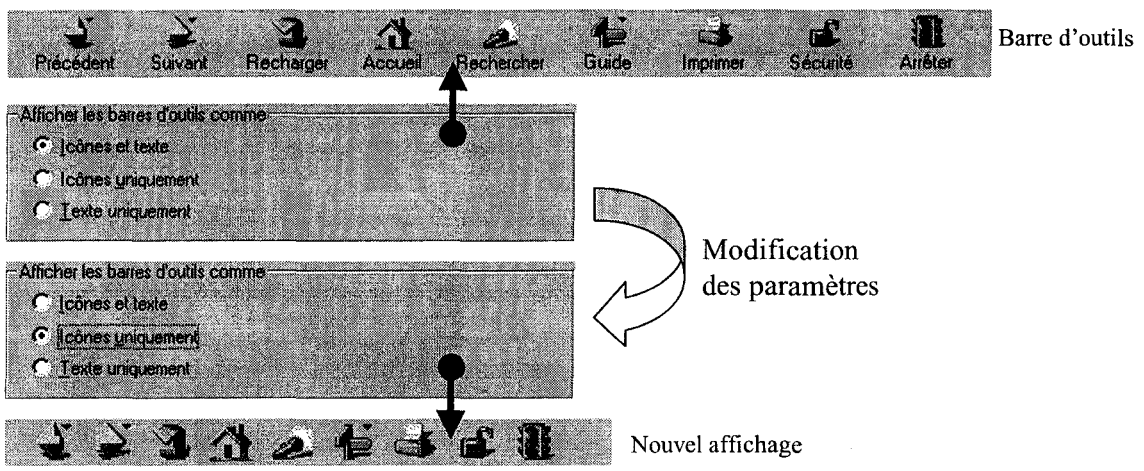

Figure 11. Paramétrage des préférences dans le navigateur Netscape : L'utilisateur choisit un paramètre parmi des choix prédéfinis.

Du point de vue de l'AT, la paramétrage permet d'adapter l'artefact médiateur à l'activité. La limite est que le développeur de l'artefact doit avoir prévu cette adaptation, ce qui implique qu'il ait anticipé certaines situations. Dans l'exemple de la Figure 11, le sujet peut choisir un affichage avec ou sans texte. L'affichage avec texte prend plus de place dans l'écran que l'affichage sans texte, ce qui réduit la zone correspondant à l'affichage du document en cours de visualisation. L'affichage avec texte permet au sujet peu expérimenté d'avoir des informations qui lui permettent d'interagir plus vite avec son navigateur. Chaque nom de fonction est un aide mémoire quant aux fonctions associées à chaque bouton. La zone de contenu s'en trouve réduite. Le sujet inexpérimenté accepte cette réduction car un équilibre s'installe entre la contrainte imposée par son écran, ses besoins en terme de fonctions directement accessibles, et ses besoins envers la zone de contenu. Au fur et a mesure de son activité, le sujet acquiert une expérience qui fait disparaître son besoin d'affichage avec texte. Son attention se porte de plus en plus sur la zone de contenu qui prend une place plus importante dans son activité. Un déséquilibre s'installe alors et un nouveau paramétrage

s'impose : spécifier une barre d'outils sans texte qui prend moins de place. L'application, par le paramétrage, peut être adaptée à différentes situations. Cependant, lorsque l'adaptation n'a pas été prévue, le sujet doit trouver un autre moyen : changer d'application par exemple.

4.2.4.2 L'intégration

L'intégration permet à l'utilisateur d'intégrer et de lier des composants. Comme le soulignent Koch et Teege [66], ce type de malléabilité permet aussi bien d'ajouter que d'enlever des composants de l'application. Un exemple aujourd'hui courant de malléabilité par intégration est celle que l'on peut trouver avec les Plug-Ins. Certains Plug-Ins pour les navigateurs Netscape leur permettent de devenir compatibles Java Swing [37], ce qui augmente les capacités et rend possible l'exécution d'Applets construites sur les JFC (Java Foundation Classes) de Java 2. Dans un domaine totalement différent, les Plug-Ins Photoshop permettent aux graphistes d'utiliser de nouveaux filtres pour travailler leurs images.

L'intégration dépend fortement du langage de composition sous-jacent. Il existe deux types d'intégration : l'intégration 'dure' (*hard integration*) et l'intégration 'douce' ou fine (*soft integration*) [88]. L'intégration dure est réalisée dans une sorte de copier/coller du composant dans l'application qui se charge de le lier au reste du système. Dans l'intégration fine, le sujet doit utiliser un langage plus évolué que le simple copier/coller (comme un langage de scripts) pour spécifier les liens entre le composant nouvellement intégré et le reste de l'application. Dans ce cas, les problèmes rejoignent ceux de la programmation par l'utilisateur : une intégration fine implique que le sujet connaisse ou apprenne un langage de composition. Toutefois, notons qu'un langage de composition peut être plus simple qu'un langage général de programmation comme java. Le langage de composition permet d'assembler des composants. De ce fait, son niveau d'abstraction est lié à celui des composants à assembler : il est plus simple d'assembler des briques que des molécules ! Stiemerling, auteur des FlexiBeans [124] et de la plate-forme EVOLVE [125], souligne de plus que le langage de composition, s'il permet de réaliser une composition hiérarchique, peut servir de base à la malléabilité hiérarchique.

L'intégration propose une solution plus générique que le paramétrage. Ici, le développeur de l'artefact n'est pas obligé d'anticiper tous les besoins des sujets. Il peut apporter de nouvelles réponses aux problèmes émergeants. Les solutions, sous formes de composants, peuvent être intégrées après le déploiement de l'application qui a donc moins de chances d'être rejetée. L'approche par l'intégration est prometteuse par rapport aux processus de développement

décrits dans l'AT. Toutefois, notons qu'elle implique encore fortement le développeur dans la construction de nouvelles solutions. Celui-ci ne peut être éliminé de l'équation que lorsque tous les composants permettant de répondre aux besoins des sujets ont été développés.

Il serait faux de penser que la tâche du concepteur est simplifiée. En effet, pour que le mécanisme d'intégration soit disponible, le concepteur se trouve face à un nouveau problème. Il ne doit plus fournir des solutions particulières à des problèmes anticipés, mais il doit proposer une solution générique pour les résoudre.

4.2.4.3 L'extension

L'extension d'une application correspond à en modifier l'implémentation. Lorsqu'une application est constituée de composants, la notion d'extension dépend du plus bas niveau d'abstraction considéré en terme de composant. Comme le souligne Stiemerling [123], la malléabilité est supportée en terme d'extension à partir du moment où le composant est considéré comme atomique par rapport à la décomposition hiérarchique de l'application.

Lorsqu'elle est réalisée par un tiers, le concepteur par exemple, l'extension rejoint le schéma intégrateur que nous venons de présenter. La malléabilité par extension telle que nous l'entendons suppose qu'elle soit réalisée par l'utilisateur final ou sujet de l'activité qui utilise ce composant. Elle apporte alors la solution idéale par rapport à l'AT. En effet, par l'extension, le sujet est réellement à même de modifier lui-même l'application pour qu'elle réponde à ses besoins émergeants. Malheureusement, c'est une vision idéaliste difficilement réalisable car la malléabilité par l'extension n'est pas un problème simple à mettre en œuvre efficacement !

L'extension implique une modification dite 'radicale' du composant qui sous-entend une programmation dans un langage bien défini. Le langage qui offre la plus grande souplesse est celui dans lequel le composant a été implémenté. Le choix du langage d'implémentation a une grande importance dans cette approche de la malléabilité. Un tel langage doit permettre d'étendre le composant sans corrompre l'ancien code [88]. Ce point argumente largement en faveur des langages orientés objets qui proposent les mécanismes adéquats au travers de l'héritage et de la spécialisation. Malheureusement, la maîtrise d'un tel langage est l'apanage de l'informaticien et non celui de l'utilisateur.

Pour l'extension, le travail du concepteur est d'implémenter les fondements de l'application et de fournir des moyens aussi compréhensibles que possible (proche du domaine problème) permettant à l'utilisateur de compléter les blancs. Par exemple, un composant pourra être vu comme une coquille qui possède certaines fonctions et propriétés génériques (ce qui fait d'elle un composant dans le cadre de l'application) et qui peut être remplie par l'utilisateur.

4.2.5 *Le prix de la malléabilité*

"The price of tailoring flexibility is paid at the expense of having to master an increased amount of computational complexity " ([90] p 76)

La malléabilité a un prix : quelle que soit la manière d'envisager la malléabilité, sa puissance croit en même temps que l'effort qui doit être fourni par l'utilisateur pour y accéder.

L'approche hiérarchique organise la malléabilité dans différents niveaux d'abstractions permettant des modifications plus fines au fur et à mesure que le niveau d'abstraction décroît. Cette approche permet à l'utilisateur de modifier l'application au niveau d'abstraction qui l'intéresse, ou qu'il est capable d'appréhender. Cependant, la malléabilité hiérarchique présuppose que l'utilisateur descende dans les niveaux pour obtenir le maximum de son application. Plus le niveau de malléabilité s'accroît, plus le niveau d'abstraction diminue. Le plus bas niveau d'abstraction envisageable est celui utilisé dans le langage d'implémentation des composants de plus bas niveau. Autrement dit, plus on veut pouvoir modifier finement l'application, plus les composants sont proches des fondements informatiques. Le travail du concepteur est d'organiser la structure hiérarchique de manière à ce que le passage d'un niveau d'abstraction à un autre soit envisageable.

De la même manière, les différents mécanismes supportant la malléabilité introduisent différents niveaux de difficulté. Morch identifie une distance entre le code d'implémentation de l'application et les objets ou composants directement présentés à l'utilisateur. Cette distance est nommée *distance de conception*. Dans une application monolithique et non malléable, la distance de conception est totalement parcourue par le concepteur. Par définition, la malléabilité considère l'utilisateur comme un concepteur partiel de sa propre application. Celui-ci doit donc parcourir une partie de la distance de conception. L'utilisateur part des éléments qu'il manipule pour descendre vers les mécanismes qui servent l'implémentation de l'application. Pour l'utilisateur, la difficulté augmente avec la distance de conception qu'il doit lui-même parcourir (cf. Figure 12).

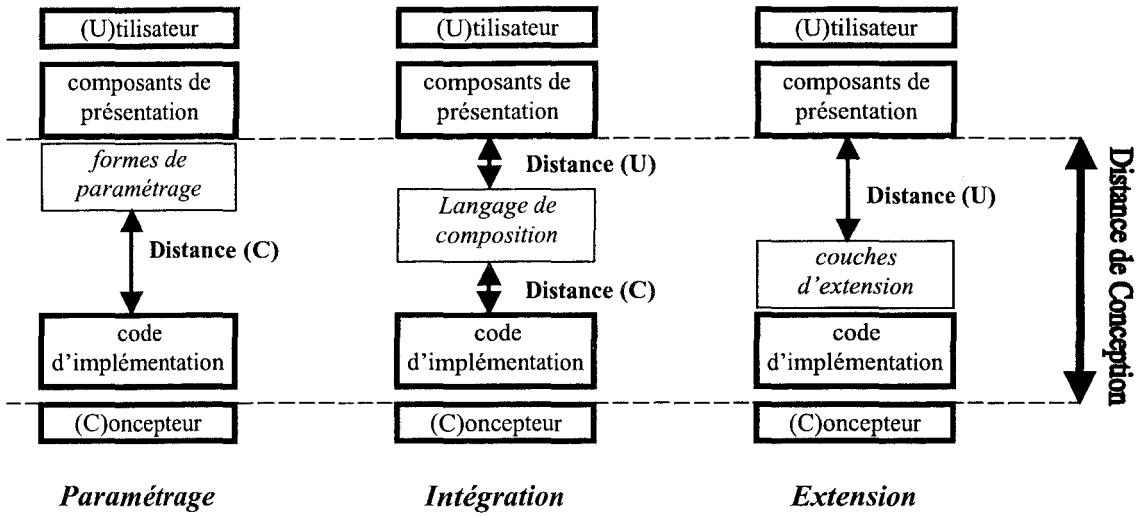

Figure 12. La malléabilité par paramétrage, par intégration et par extension.
L'effort fourni par l'utilisateur croît avec le degré de malléabilité.
(repris et adapté de [90], p 42)

Le paramétrage est le mécanisme le plus simple à appréhender car il correspond à spécifier des propriétés des composants manipulés. Avec le paramétrage, la structure composants de l'application peut même être transparente pour l'utilisateur qui peut ne pas avoir conscience de paramétrer des composants, mais penser directement paramétrer l'application. Avec le paramétrage, la distance de conception est pratiquement totalement parcourue par le concepteur qui tente de s'approcher au plus près des problèmes pouvant être rencontrés par l'utilisateur.

L'intégration demande plus d'effort de compréhension de la part de l'utilisateur car celui-ci doit non seulement comprendre comment le composant est utilisable (un éventuel paramétrage), mais surtout, comment il va interagir avec le reste de l'application. Dans le cas d'une intégration fine, la difficulté est encore plus grande car l'utilisateur doit lui-même créer certains des liens entre les composants. Avec l'intégration, le concepteur laisse une partie de la conception à la charge de l'utilisateur.

Enfin, la plus grande difficulté pour l'utilisateur se situe au niveau de l'extension. Ici, les composants ou les mécanismes qui servent l'application peuvent potentiellement être créés et redéfinis par l'utilisateur. Dans ce cas de figure, l'utilisateur se rapproche tellement de l'implémentation qu'il est difficile d'en dissimuler les fondements informatiques. La distance

à parcourir pour l'utilisateur est telle qu'il est pratiquement au même niveau que le concepteur.

Nous voyons que la malléabilité croît avec l'effort qui doit être fourni par l'utilisateur pour la maîtriser. Ni l'approche hiérarchique, ni les différents mécanismes suggérés ne permettent d'échapper à cette règle. A l'inverse, plus la malléabilité est grande, plus elle semble répondre aux besoins des utilisateurs. A l'exposé du prix de la malléabilité, la question est : l'utilisateur sera-t-il enclin à la mettre en pratique ? Si l'on se replace dans le cadre de l'AT ou de DARE, la malléabilité est nécessaire car elle permet au sujet de spécifier ses besoins émergeants en cours d'activité. Rappelons que l'objet de l'activité *motive* le sujet quant à la réalisation de sa tâche. Le sujet n'utilise la malléabilité que lorsqu'une panne intervient dans l'activité. Nous pensons que si la malléabilité est en adéquation avec les éléments qui sont susceptibles de causer de telles pannes, le sujet sera assez motivé pour apprendre à adapter son application. La motivation permettra de fournir l'effort nécessaire à l'augmentation de l'expérience envers la malléabilité du système. *Il faut qu'un équilibre puisse s'installer entre la motivation de l'utilisateur pour achever son activité et l'effort à fournir pour maîtriser l'application.*

Notons toutefois que cet effort peut être réduit du fait qu'un utilisateur n'est généralement pas le seul sujet de l'activité. Comme le soulignent Mackay [76] et Nardi [93], il arrive souvent dans une communauté qu'un ou plusieurs sujets jouent un rôle particulier ('translator' pour Mackay, 'gardener' pour Nardi) de par leur grande expertise et/ou leur facilité à utiliser les mécanismes complexes d'un système (les mêmes remarques ont été faites par [77][58]). Ces sujets aident les sujets moins experts, ce qui a pour effet de débloquer l'activité. De plus, cette aide peut avoir pour conséquence d'augmenter l'expertise des sujets qui en ont besoin. Ainsi, la présence de sujets experts dans une activité a pour effet de faciliter l'équilibre entre la motivation du sujet pour achever son activité et l'effort qu'il doit fournir pour adapter l'application. Néanmoins, *le problème du concepteur est d'offrir des moyens adéquats qui permettent à un équilibre de s'installer.*

4.2.6 Un problème à deux dimensions

Comme nous venons de l'exposer, différents moyens ont été identifiés pour envisager la malléabilité. Chacun de ces moyens permet de rendre une application plus ou moins malléable. Cependant, nous avons aussi montré que la malléabilité a un prix pour l'utilisateur. Les deux principaux facteurs de difficulté sont le niveau d'abstraction des composants et les mécanismes mis en jeu.

Les niveaux d'abstraction qui peuvent être envisagés dépendent de la malléabilité hiérarchique proposée. Il faut en particulier que ces niveaux soient accessibles à l'utilisateur, non seulement fonctionnellement, mais aussi conceptuellement. Pour reprendre l'exemple de la calculette, si le plus haut niveau d'abstraction offert pour la malléabilité est celui qui concerne les composants MVC, il est peu probable que le comptable sera capable d'agir sur la conception de son application. Le problème de l'abstraction vient se greffer sur celui des mécanismes mis en œuvre pour modifier l'application. Nous avons montré que le paramétrage est le moyen le plus simple pour l'utilisateur. Cependant, paramétrer un composant de très bas niveau n'est pas une chose facile. Sous Windows par exemple, il est possible de paramétrer l'affichage avec différentes polices de caractères. Ce type de paramétrage est assez intuitif car il concerne des éléments directement accessible par l'utilisateur. De la même manière, il est possible de paramétrer la fréquence de rafraîchissement de l'écran qui agit elle aussi sur le confort de visualisation. Ceci, implique pourtant de savoir ce qu'est une fréquence de rafraîchissement, de connaître les capacités de l'écran, de la carte vidéo... Ce sont deux types de paramètres qui s'adressent à des niveaux d'abstraction différents. Autrement dit, ce n'est pas parce que le mécanisme proposé pour la malléabilité est simple, que la malléabilité l'est.

De la même manière, un mécanisme comme l'extension peut rendre la malléabilité inutilisable, même pour des composants de très haut niveau d'abstraction. Par exemple, fournir au comptable la calculette, ainsi que son code source et un compilateur java offre la plus grande malléabilité imaginable. La calculette est bien du niveau d'abstraction de l'utilisateur. Il est toutefois peu probable que l'utilisateur modifie son application, même s'il en a réellement besoin. Ce n'est pas parce que le niveau d'abstraction est conceptuellement accessible à l'utilisateur que n'importe quel mécanisme permet la malléabilité.

La malléabilité impose au concepteur d'apporter des solutions génériques non triviales devant à la fois faire converger les niveaux d'abstraction et les mécanismes adéquats aux problèmes potentiels des utilisateurs dans des situations données. En d'autres termes, le concepteur doit fournir une structure réutilisable qui spécifie les fondements de l'application malléable, les niveaux d'abstraction, la place des composants pouvant être modifiés, ainsi que les mécanismes qui y sont associés. Une telle structure peut être supportée en terme d'*architecture cadre* (ou *framework* en anglais).

4.3 Les architectures cadres

"A framework is simply a collection of several single components with predefined cooperations between them, for the purpose of accomplishing a certain task" [106]

4.3.1 L'approche Objet

Dans le domaine de l'ingénierie logicielle, la notion d'architecture cadre est apparue suite aux problèmes posés par la réutilisation logicielle. Le paradigme Objet est une base très puissante pour la réutilisation, en particulier parce qu'il supporte la conception modulaire et la programmation par différence, grâce aux mécanismes d'héritage et de spécialisation. Cependant, un système informatique est plus qu'un ensemble d'objets composants. C'est aussi une structure qui décrit les liens entre ceux-ci. La réutilisation logicielle n'envisageant un système informatique que comme un ensemble d'objets égaux pouvant être spécialisés dans un langage objet général possède le défaut de noyer la structure du système dans l'ensemble de ses composants. Cette structure constitue généralement un sous-ensemble qui fonde le système et qui peut être réutilisé.

L'approche par les architectures cadres propose de marquer explicitement la différence entre les éléments qui composent la structure réutilisable du système informatique, et les éléments ou composants qui peuvent être modifiés, étendus ou remplacés. Les architectures cadres peuvent aussi servir de base au développement du système par des utilisateurs moins spécialisés puisqu'ils n'ont plus à se soucier de sa structure profonde, mais peuvent focaliser sur les composants qui les intéressent. Dans une architecture cadre, on ne réutilise pas que du code, mais toute une architecture logicielle dédiée à une certaine tâche générique. En d'autres termes, une architecture cadre permet de réutiliser à la fois le code et la conception. Il est à noter que les architectures cadres sont souvent conçues à partir d'un patron. Les patrons s'adressent plus spécifiquement aux concepteurs. Un patron est une solution à une classe de problèmes. Une architecture cadre est une implémentation de cette solution. De ce fait, une architecture cadre s'adresse plus spécifiquement aux développeurs.

Dans une architecture cadre certains des composants sont figés. Ils participent à la structure architecturale de l'architecture cadre et ne doivent donc pas bouger. Ces aspects fixes qui sont communs à toutes les applications du domaine adressé par l'architecture cadre sont nommés les *points gelés (frozen spots)* ou *points froids* [47][105]. D'autres composants peuvent être modifiés, remplacés ou réarrangés pour fournir une solution à un problème particulier du

domaine. Les points variables ou de modification sont appelés les *points chauds (hot spots)* [47][105]. Une architecture cadre peut être définie selon le domaine dans lequel elle est utilisée ainsi que par les moyens à mettre en œuvre pour la spécialiser.

La classification par le domaine identifie principalement deux types d'architectures cadres : *les architectures cadres d'applications génériques* et *les architectures cadres dédiés à un domaine spécifique*. Les architectures cadres d'application générique fournissent des fonctionnalités basiques communes à tous les programmes. Des exemples bien connus de telles architectures cadres sont MVC (Model View Controller) ou MFC (Microsoft Foundation Classes). Les architectures cadres dédiées à un domaine spécifique fournissent des fonctionnalités adaptées à un domaine de problèmes spécifiques. Ce sont par exemple des architectures cadres dédiées au TCAO [56][66][125].

La classification par les moyens à mettre en œuvre pour spécialiser une architecture cadre identifie elle aussi deux types d'architectures cadres : les *boîtes blanches* (*white-box framework*), et les *boîtes noires* (*black-box framework*) [43].

Dans les architectures cadres *boîtes blanches* à objets, les points chauds correspondent à des classes qui peuvent être modifiées ou remplacées. Il est possible de remplacer une classe par une autre se trouvant dans la même hiérarchie. Les points chauds sont généralement implémentés sous formes de classes abstraites. Les moyens utilisés sont un langage de programmation objet, l'héritage et la spécialisation. Cette approche oblige l'utilisateur de l'architecture cadre à comprendre la hiérarchie de classes existante pour pouvoir dériver de nouvelles classes qui doivent être réinsérées dans l'architecture. Apprendre à utiliser une architecture cadre boîte blanche signifie comprendre comment elle est construite. L'utilisation d'une architecture cadre boîte blanche correspond à faire de la malléabilité par extension.

Les architectures cadres *boîtes noires* utilisent la composition d'objets. Les composants sont généralement organisés dans une librairie de composants. Chaque composant utilise un protocole particulier. L'utilisateur d'une architecture cadre boîte noire n'as pas besoin de connaître l'intérieur des composants qu'elle utilise. Il ne doit en comprendre que l'interface externe, d'où l'abstraction de boîte noire. L'utilisation d'une architecture cadre boîte noire pure correspond à faire de la malléabilité par intégration.

Les architectures cadres boîtes noires sont moins flexibles que les architectures cadres boîtes blanches car le nombre de combinaisons possibles est déterminé par le nombre de composants utilisables et prédéfinis. Par contre, le bénéfice est qu'elles demandent moins de

programmation car les composants sont simplement combinés et non pas étendus par héritage/spécialisation. Hummes et Merialdo [56] soulignent que les applications qui doivent être étendues à l'exécution doivent reposer sur des architectures cadres boîtes noires car elles anticipent les extensions en définissant des interfaces et points d'entrée qui permettent d'insérer de nouveaux objets. Il est possible, à l'exécution, de remplacer un composant par un autre qui implémente le même protocole. Nous ne sommes pas tout à fait en accord avec le point de vue de Hummes et Merialdo car, en utilisant la programmation réflexive, il est tout aussi possible de modifier une architecture cadre boîte blanche à l'exécution. Toutefois, nous admettons que la mise en place de tels mécanismes n'est pas simple. Enfin, pour palier aux problèmes de la limitation du nombre de composants d'une architecture cadre boîte noire, il est possible de la coupler avec les fonctionnalités d'une architecture cadre boîte blanche. Ceci permet aux utilisateurs expérimentés d'en augmenter les capacités en définissant de nouveaux composants réutilisables.

Un problème majeur, dans la création d'une architecture cadre, est de définir les points chauds. Une architecture cadre bien définie identifie des points chauds orientés domaine [106]. Puisque l'architecture cadre est créée pour fournir une architecture générique pour une tâche générique, le domaine correspond à une tâche plus ou moins abstraite. Le problème est d'identifier ce qui rend cette tâche abstraite de manière à identifier les points chauds qui seront utilisés dans la construction d'une solution concrète. Le processus idéal de développement d'une architecture cadre commence par une analyse. Suivent alors la conception puis l'implémentation. Enfin, une phase de test permet de vérifier si l'architecture cadre est assez générique, bien conçu, etc. Le problème principal se situe dans la phase d'analyse. Il est très difficile et long d'effectuer une bonne analyse à partir du domaine problème dans le but d'abstraire les éléments importants qui servent à la conception de l'architecture cadre. Une démarche possible est celle qui tire des leçons de résultats empiriques. C'est celle qui est la plus employée dans le domaine du TCAO.

4.3.2 L'approche composants dans le domaine du TCAO

L'évolution des systèmes pour le TCAO vers les approches composants a aussi eu pour effet d'introduire les architectures cadres dans notre domaine de recherche. Les architectures cadres telles que MVC ou CORBA sont bien entendu utilisées pour développer les collecticiels, mais elles ne le sont pas exclusivement pour le TCAO. Les architectures cadres du TCAO sont généralement des architectures cadres dédiées à un domaine spécifique

souvent plus précis que le domaines du TCAO lui-même [56][66][125][52]. L'approche de l'architecture cadre, permettant de construire une réalisation ouverte basée sur les composants, est une solution efficace au support de la malléabilité.

Toutefois, on peut remarquer que la plupart des architectures cadres pour le TCAO sont des boîtes noires. La particularité de la malléabilité est que le système doit être modifiable par l'utilisateur final. Si possible, la malléabilité doit même être proposée à l'exécution du système et les modifications ne doivent pas impliquer un redémarrage intempestif. Autrement dit, le problème du TCAO est de fournir une application qui repose sur une architecture cadre accessible aux utilisateurs finaux, à l'exécution, et souvent au travers de l'application elle-même. Comme nous l'avons déjà exposé, les architectures cadres boîtes noires sont d'une utilisation plus facile que les architectures cadres boîtes blanches. Elles requièrent moins de compétences en terme de programmation. De plus, nous avons vu que les architectures cadres boîtes noires sont, à priori, plus aisément modifiables au cours de leur propre exécution. Ceci explique certainement la préférence du TCAO pour les architectures cadres boîtes noires.

L'utilisation d'une architecture cadre boîte noire ne permet, en terme de malléabilité, que de faire de l'intégration et du paramétrage de composants (suivant sa réalisation, un composant peut être paramétré au moment de son intégration et/ou au cours de son utilisation). Il est plus simple de faire de l'intégration que de faire de l'extension. Cependant, l'extension offre une malléabilité plus puissante. En particulier, lorsqu'un composant n'existe pas dans l'architecture cadre boîte noire et que ce problème doit être réglé en terme de malléabilité (ce n'est pas un développeur externe qui modifie l'architecture cadre), la malléabilité par extension est nécessaire.

Il existe plusieurs problèmes récurrents dans la création d'architectures cadres pour le TCAO. Un premier problème est celui de l'interface permettant d'accéder à l'architecture cadre. En effet, dans les architectures cadres à objets, l'interface est bien souvent un langage de programmation général tel Smalltalk. Dans des architectures cadres permettant la création d'interfaces graphiques, cette interface est souvent d'un niveau d'abstraction plus élevé, mais elle s'adresse tout de même plus spécifiquement à des développeurs qu'à des utilisateurs finaux. Le problème du TCAO est qu'il doit offrir une architecture cadre pour des activités coopératives orientées vers un certain domaine proche de l'utilisateur final, et que l'accès à la malléabilité doit être compréhensible dans le cadre de ce domaine. En d'autres termes, l'utilisateur qui manipule des documents hyper-textes ne doit pas tout à coup se retrouver en train de manipuler des instances javax.swing.Jframe (les objets fenêtres de Java 2). Même si

c'est le cas, en terme d'implémentation, il doit exister une couche intermédiaire qui crée un lien entre l'objet de présentation et l'objet d'implémentation. Il en est de même pour le langage d'intégration ou d'extension. L'ensemble doit bien entendu être présenté à l'utilisateur au travers d'une interface homme–machine adaptée.

Un autre problème récurrent est d'identifier les points chauds de l'architecture cadre. Notons que l'identification des points chauds fournira les bases d'une éventuelle malléabilité hiérarchique. La question qui se pose alors est : quelles sont les abstractions qui serviront de points chauds ? Dans la littérature du TCAO, les auteurs soulignent généralement cette difficulté et proposent d'affiner la définition des points chauds au cours du test de l'architecture cadre dans la situation particulière où elle va être utilisée. Notre démarche, dans la conception de DARE, est quelque peu différente.

4.4 DARE et la malléabilité : une première approche

L'AT se veut elle-même un cadre conceptuel des sciences humaines. Elle a été conçue à partir des résultats de nombreuses et diverses recherches sur l'activité humaine. Notre but étant de fournir les fondements d'un système permettant de supporter cette activité, l'idée de base de DARE est de créer une architecture cadre informatique directement inspirée du cadre conceptuel proposé par l'AT. Celle-ci nous permettra d'identifier les points chauds de l'activité. La structure basique de l'activité d'Engeström fournit une base fort appréciable pour la construction d'une telle architecture cadre. De plus, selon l'AT, tous les points chauds peuvent être construits et/ou étendus par les sujets au cours de leur activité. Si l'on se réfère à cette propriété de l'activité humaine, la malléabilité par paramétrage ou par intégration ne suffit pas. Chaque élément, chaque composant de l'activité doit pouvoir être étendu ! Le problème est d'identifier des moyens permettant aux sujets de trouver un équilibre entre la malléabilité qui leur est nécessaire et sa difficulté d'appréhension.

DARE doit donc être un système malléable. Pour cette malléabilité, nous pensons qu'il est nécessaire de permettre aux sujets d'être en harmonie avec leurs besoins, leur expérience et leur motivation. Rappelons, que l'expérience des sujets envers leur activité est cristallisée dans les éléments qu'ils construisent pour la réaliser. Ici, ces éléments sont les composants. Ceci nous laisse espérer qu'une architecture cadre boîte blanche, par l'extension de composants, permettra aux sujets de construire eux-mêmes les briques d'une architecture cadre boîte noire plus aisément utilisable. C'est pourquoi, pour DARE, nous opterons vers une malléabilité à la fois hiérarchique, par paramétrage, par intégration et par extension en

mettant en place une architecture cadre qui sera à la fois boîte blanche et boîte noire. Notre espoir, en permettant de cristalliser l'expérience des sujets au sein des composants étendus, est que la partie boîte blanche de l'architecture cadre (l'extension) ne soit petit à petit plus utilisée : tous les composants basiques ayant été construits par les sujets eux-mêmes dans diverses activités coopératives.

4.5 Résumé sur la malléabilité envisagée pour notre environnement de TCAO

Le Tableau 5 résume les différents points que nous retenons dans notre approche de la malléabilité pour DARE.

Un équilibre doit pouvoir s'installer entre la motivation du sujet, la difficulté d'utilisation de la malléabilité, et le niveau de malléabilité proposé. Pour ce faire :

- DARE met en œuvre des composants qui facilitent la *réutilisation* logicielle.
- DARE repose sur une architecture cadre boîte noire : elle permet *l'intégration*, éventuellement dans une structure *hiérarchique* à différents niveaux d'abstraction.
- DARE repose sur une architecture cadre boîte blanche : elle permet *l'extension*, et offre le niveau de malléabilité maximal.
- DARE repose sur une architecture cadre dédiée à un domaine spécifique, proche du sujet, et *défini par le cadre conceptuel de la Théorie de l'Activité.*

Tableau 5. La malléabilité dans DARE

Ayant présenté les moyens, en terme d'ingénierie logicielle, que nous avons identifiés pour réaliser un système malléable, il nous faut maintenant définir le modèle conceptuel de ce système. Ce modèle servira à construire l'architecture cadre logicielle de DARE.

Chapitre 5

Le modèle conceptuel du support à l'activité

La modélisation conceptuelle du support à l'activité est pour DARE une phase très importante. Le modèle proposé doit être assez générique pour permettre de supporter toutes les activités désirées par les sujets. Rappelons simplement qu'il est à priori impossible de prévoir le modèle spécialisé qui sera mis en œuvre pour supporter une activité particulière puisque celui-ci sera construit au fur et à mesure de son exécution.

Le modèle conceptuel que nous proposons diffère du modèle hiérarchique d'ODESCA présenté dans le chapitre 1. Il est le fruit de notre démarche générale exposée dans le chapitre 2 et tire partie des résultats de notre étude de la Théorie de l'Activité (AT) dans le chapitre 3. Il doit identifier les concepts et mécanismes sous-jacents qui permettront à notre environnement de Travail Coopératif Assisté par Ordinateur (TCAO) de fournir des supports à l'activité coopérative malléables, selon la définition de la malléabilité donnée au chapitre 4.

En particulier, le modèle proposé doit être compréhensible par les sujets. Ceux-ci doivent être capables d'analyser un support d'activité à un instant donné pour être en mesure de le faire évoluer en fonction des besoins émergeants. De plus, la construction ou l'évolution d'un support d'activité par les sujets implique qu'ils comprennent les mécanismes permettant de modifier un modèle particulier dans le cadre du modèle générique que nous proposons. Enfin, ce modèle générique ainsi que les mécanismes qui l'accompagnent doivent rester compatibles avec ceux de l'AT qui décrit les fondements de l'activité humaine. De la même manière, il nous faut identifier des liens fins avec des stratégies de conception logicielle car, dans DARE, le sujet est le concepteur et développeur de son propre environnement de TCAO. Pour construire ce modèle et mettre en place les mécanismes requis à la réalisation de nos enjeux, nous nous sommes basés sur la structure basique de l'activité proposée par Engeström.

5.1 Retour sur la structure basique de l'activité

5.1.1 Remarques préliminaires

La structure basique de l'activité a été présentée dans le chapitre 3. De notre point de vue, ce modèle possède l'avantage d'être générique dans le sens où il permet de décrire toute activité. De plus, dans son ensemble, l'AT expose des propriétés et mécanismes de l'activité humaine qui peuvent être appliqués ou compris dans le cadre du modèle d'Engeström comme, par exemple, la construction de règles qui influencent les sujets au sein d'une communauté.

Nous aimerions cependant introduire quelques remarques préliminaires à la construction du modèle générique de DARE. Chaque fois que la structure d'Engeström est présentée, les relations entre les différents éléments qui la composent sont bien explicitées alors que la définition des éléments eux-mêmes reste assez floue. L'outil par exemple est généralement défini comme tout ce qui est utilisé dans le processus de transformation, aussi bien les outils matériels que les outils pour penser. Le concept de règle est lui aussi assez peu défini. Les règles sont des lois, un ensemble de pratiques communes acceptées au sein de la communauté. Les règles peuvent être implicites ou explicites. La question est : comment ou sous quelle forme les règles sont-elles explicitées ?

Le manque de définition précise est pour nous un indicateur sur le but des auteurs. L'objectif ici ne semble pas être d'exposer en détail ce que sont les éléments apparaissant dans la structure d'Engeström, mais plutôt d'étudier les mécanismes environnants qui les influencent. Par exemple, il ne s'agit pas de décrire ce que sont les règles mais d'expliquer pourquoi et comment elles évoluent au cours de l'activité. Certaines propriétés sont cependant mises à nu, comme le fait que chaque élément médiateur soit capable de cristalliser une partie de l'héritage historique et culturel de la situation, mais le détail de l'élément est laissé de côté. De cette manière, l'approche d'AT est générique, exposant les mécanismes communs à toute activité. Celui qui désire utiliser l'AT pour analyser une activité particulière peut ainsi lui-même spécialiser chacun des éléments pour modéliser la situation qui l'intéresse.

Notre conclusion est que l'AT expose des mécanismes génériques indépendants de la spécificité des éléments mis en jeu dans une activité particulière. Ceci se révèle d'un grande importance pour notre réalisation dans le cadre de DARE. Il nous faut fournir un support informatique aux différents éléments du support à l'activité. Un support informatique signifie inévitablement une explicitation de ces éléments. La littérature que nous avons pu étudier ne

fournit pas d'information particulière concernant une telle explicitation pour chacun des éléments du modèle. Il nous a donc fallu faire des choix quant aux détails de notre modélisation. Cependant, l'analyse que nous avons brièvement exposée ici nous montre que pour répondre aux besoins de DARE, dans un premier temps, l'intérêt ne doit pas tant être porté sur la modélisation des éléments eux-mêmes, mais précisément sur les mécanismes et propriétés génériques qui les caractérisent. Ainsi, dans le cadre de cette thèse, nous nous sommes principalement focalisés sur les moyens à mettre en œuvre pour permettre aux sujets utilisant DARE de faire évoluer le support à l'activité coopérative, plutôt que sur la réalisation des éléments eux-mêmes. A l'image de l'AT, qui propose un cadre conceptuel générique pour l'étude d'activités particulières, DARE propose une architecture générique mettant en œuvre les concepts et mécanismes fondamentaux qui nous semblent nécessaires au support de toute activité coopérative.

5.1.2 Rappel des éléments de la structure basique de l'activité

Les éléments à la base de notre modèle conceptuel sont l'objet (le motif de l'activité), le sujet (car l'objet ne peut être réalisé s'il n'y a pas de sujet qui y travaille), la communauté (un ensemble de sujets partageant le même objet d'activité), l'outil (médiatise la relation sujet-objet ; tout ce qui est utilisé dans le processus de transformation), les règles (médiatisent la relation sujet-communauté ; lois explicites et implicites, pratiques acceptées...), et la division du travail (médiatise la relation communauté-objet ; organisation explicite et implicite de la communauté). Il est important de rappeler que chacun de ces éléments évolue au cours de l'activité et porte en lui une partie de l'histoire et l'héritage culturel de la relation qu'il médiatise.

5.2 Introduction de concepts complémentaires

5.2.1 Le concept de rôle

Pour notre conception, nous avons choisi d'introduire le concept de *rôle* et d'y synthétiser les concepts de règles et de division du travail. Ce choix est polymotivé.

Premièrement, les utilisateurs de collecticiels, et plus généralement, les utilisateurs de systèmes informatiques, sont habitués à la notion de rôle. Sous Windows NT par exemple,

l'identification peut être faite en tant que simple *utilisateur* ou en tant qu'*administrateur*. Qui n'a jamais rêvé d'être le *super utilisateur* sous Unix ou Linux ? La vie réelle n'est elle-même que situations et jeux de rôles : le patron et l'ouvrier, le professeur et l'élève. La notion de rôle est intuitive car nous la manipulons chaque jour.

Secondement, nous pensons qu'il est plus aisé pour un sujet de comprendre son rôle que de découvrir un ensemble de règles générales ainsi qu'une division du travail. Dans une entreprise, il n'est pas immédiatement nécessaire de connaître exactement ce que font les autres pour remplir sa part dans la division du travail. Par contre, dans le cas où un sujet a besoin de plus d'information, il doit pouvoir découvrir le rôle des autres, ce qui lui permet de créer l'articulation entre son propre rôle et le reste du monde. Toutefois, nous admettons qu'il peut être difficile de reconstruire l'ensemble des règles ainsi que la totale division du travail à partir d'un jeu de rôles.

N'oublions pas qu'un de nos enjeux, à terme, est de permettre aux sujets de spécifier eux-mêmes les éléments qui composent leur support d'activité, et donc leurs rôles. Ceci revient à modéliser une partie de l'activité. Or, comme nous le verrons plus avant dans cette thèse (cf. chapitre 8), l'approche de la modélisation par les rôles semble être la plus intuitive. Celle-ci est même utilisée pour modéliser l'activité d'objets informatiques dans le domaine de la conception logicielle [112][67]. C'est ainsi qu'on a pu voir apparaître récemment des méthodes de modélisation telles que OOram [109]. Cette nouvelle approche est prise en compte jusque dans UML [111][92], le langage de modélisation unifié de l'Object Management Group (OMG).

Le concept de rôle apparaît très peu dans l'AT. Il n'est mentionné que pour définir d'une manière générale ce qu'est la division du travail. Dans cette optique, le rôle ne correspond qu'à la place tenue par un sujet dans la communauté par rapport à ce qui doit être fait pour réaliser le travail en cours. Il ne médiatise donc que la relation communauté-objet. Le rôle d'AT introduit des devoirs pour le sujet qui le possède. Nous pensons que la notion de devoir est souvent liée à la notion de droit. Un sujet jouant un certain rôle réalise une partie du travail. Il doit donc posséder les droits nécessaires sur les éléments de l'activité qui lui permettront de tenir cette place dans la communauté. Par exemple, un administrateur de système informatique possède un grand nombre de droits parce qu'il a le devoir de maintenir le système en bon état. Inversement, les sujets jouant d'autres rôles ne doivent idéalement pas posséder des droits pouvant interférer dans cette décomposition du travail.

Dans l'AT, les droits sont définis par les règles en vigueur. Dans le modèle générique de DARE, le rôle inclut donc aussi le concept de règles et médiatise aussi la relation sujet-communauté. Il détermine exactement ce que signifie être un membre de la communauté. Les règles qu'il représente sont plus ou moins explicites mais influencent fortement les actions qu'un sujet le jouant peut exécuter au cours de l'activité.

La Figure 13 est une représentation graphique des concepts qui ont été exposés, ainsi que leurs principales interrelations. Elle constitue la vision de la structure l'Engeström telle qu'elle apparaît dans DARE. La seule différence consiste en une synthèse des concepts de règles et de division du travail. Cependant, nous voulons insister sur le fait que cette synthèse ne renie en aucun cas les mécanismes et fondements proposés par l'AT. Un jeu de rôles présent dans une activité représente bien des règles et une division du travail qui l'influencent. L'évolution d'un jeu de rôle, ou d'un rôle particulier, correspond à une évolution des droits et devoirs sous-jacents. Ainsi, pour conserver la cohérence avec l'AT, dans DARE, les deux facettes du rôle doivent pouvoir être modifiés par les sujets, en fonction des besoins émergeants. L'outil et le rôle sont les éléments ou artefacts médiateurs de l'activité.

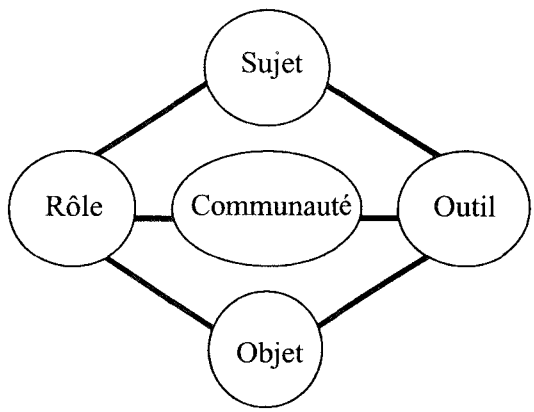

Figure 13. Structure d'une activité dans DARE

5.2.2 *Le concept de tâche*

Nous avons identifié les éléments qui entrent en jeu dans le support à l'activité coopérative offert par DARE. L'environnement gère un ensemble d'outils et de rôles. A priori, ceci ne semble pas très révolutionnaire pour un collecticiel ! L'originalité de notre approche est de mettre en œuvre les mécanismes proposés par l'AT qui permettent, entre autre, de modifier ces rôles et ces outils au cours de l'activité dans laquelle ils sont impliqués, ce qui correspond

à faire évoluer le support à l'activité en cours d'exécution. Cependant, une question subsiste : qu'est-ce que spécifier un support d'activité ?

Du point de vue de l'informaticien, ceci revient à modifier l'application et donc altérer son modèle d'exécution, c'est-à-dire son programme. Même si la modification d'une application en cours d'exécution n'est déjà pas en soi un problème simple, il est bien entendu peu souhaitable de demander aux sujets voulant spécifier de nouveaux besoins de reprogrammer leur collecticiel. A priori, le concept de modèle d'exécution et le langage de programmation utilisé ne leur sont pas familiers. Dans le cas contraire, l'utilisateur de DARE serait bien plus qu'un 'simple' sujet !

Pour que les modifications puissent être réalisées de manière intuitive et dans le contexte unificateur qu'est l'activité, il nous faut identifier un concept représentatif pour la spécification du support d'activité. Une fois ce concept identifié, il restera encore à définir un langage et les mécanismes qui permettent aux utilisateurs de modifier le support d'activité au cours de sa propre exécution.

Bedny, dans son livre *The Russian Theory of Activity* [3] mentionne que la tâche est le composant de base de l'activité. Nos vies peuvent être conceptualisées comme une succession continue de tentatives de réalisation de tâches variées. L'auteur souligne que le développement de systèmes, y compris les systèmes informatiques, commence souvent par l'analyse et la conception d'une tâche. Le mot tâche possède malheureusement un sens polysémique pouvant mener à l'incompréhension et à des conflits entre ceux qui l'utilisent[1]. Pour notre part, nous utilisons la définition de la tâche proposée par Leont'ev :

"Task is a situation requiring realisation of a goal in specific conditions" [73]

Bedny souligne que dans toute tâche, il est possible de distinguer et d'extraire des besoins et des conditions. De notre point de vue, et pour le modèle de support d'activité de DARE, les besoins sont ceux exprimés par les sujets, incluant entre autres l'objet à atteindre et les conditions qui influencent les actions de chaque sujet au cours de l'activité. L'activité correspond à la réalisation de la tâche.

L'AT explique que la situation n'est jamais donnée à l'avance et qu'elle est le résultat d'interactions, c'est-à-dire de l'activité elle-même. On pourrait donc penser à un paradoxe car, par définition, la tâche préexiste à l'activité. En réalité, ceci souligne la différence qui

[1] En particulier entre la communauté des chercheurs en interface homme-machine et celle des ergonomes.

existe entre la tâche prescrite, celle qu'on voudrait ou doit réaliser, et la tâche qui a réellement été exécutée une fois l'activité terminée. En d'autres termes, il n'existe bien une situation donnée au début et décrite dans la tâche, mais cette situation va évoluer de façon imprédictible au cours de la réalisation de cette tâche, c'est-à-dire au cours de l'activité. L'activité débute avec une situation initiale donnée dans la tâche, cette situation est partagée par les sujets impliqués, elle est transformée par le processus de réalisation de la tâche, et conclut dans un état final.

Pour notre conception, nous acceptons que le support à l'activité offert par DARE est créé selon une tâche donnée. Puisque DARE ne supporte qu'une partie de l'activité, la tâche dont nous parlons ici est bien entendu la tâche du système pour un support d'activité précis. La tâche réelle qui correspond à l'activité globale, qu'elle soit implicite ou explicite, décrit une situation plus grande dont une des conditions est d'ailleurs l'utilisation du collecticiel. Lorsque nous emploierons le mot tâche, nous parlerons donc ici de la tâche restreinte et explicite qui décrit la situation qui doit être mise en place par DARE pour une communauté particulière.

Dans DARE, la tâche définit l'objet de l'activité, les outils qui vont être utilisés dans l'environnement de TCAO, les rôles explicites qui pourront être donnés aux sujets/utilisateurs du système, ainsi que des liens avec d'autres tâches que nous allons décrire dans ce qui suit. De cette manière, la tâche est le concept unificateur recherché qui permet de décrire un support d'activité dans son propre contexte. Au lieu de programmer le système pour que celui-ci crée la situation dont il a besoin, le sujet spécifie la tâche qu'il veut accomplir, laissant à la charge de DARE de générer la situation décrite. Toutefois, il existe une différence entre le concept de tâche de l'AT et celui de DARE. Dans l'AT, la tâche spécifiée au commencement de l'activité est la tâche prescrite. Nous avons vu que l'activité a toujours tendance à en dévier. Dans DARE, cette déviation s'effectue en modifiant la tâche elle-même. Ainsi, la tâche du système correspond toujours à son activité. A notre avis, ceci possède au moins un avantage : tout nouveau sujet dans la communauté est en mesure de connaître exactement ce qui se passe dans l'environnement de TCAO, à tout instant, en consultant la tâche.

Pour conclure, retenons que dans DARE, l'activité du système se déroule selon une tâche qu'elle ne cesse de modifier jusqu'à son achèvement. Notons que, de ce fait, *la tâche est l'élément du plus haut niveau d'abstraction qui cristallise l'héritage historique et culturel de la situation*. Elle contient la description de l'ensemble des outils utilisés dans l'activité, ainsi

que les différents rôles qui ont été spécifiés pour sa réalisation. De cette manière et à un instant donné, la tâche représente un patron d'activité qui, comme nous l'avons précisé dans nos enjeux (cf. chapitre 3), pourra être sauvegardé et réutilisé car il contient l'expérience d'une communauté pour un certain objet d'activité.

5.2.3 Le concept de sous-tâche

Comme nous l'avons vu dans le Chapitre 3 (1.5.3), Kuutti utilise l'activité comme élément de base pour modéliser les organisations. Une organisation est formée d'un réseau d'activités interconnectées.

Les liens que nous proposons de supporter dans DARE sont engendrés par la division du travail. Dans notre conception, le rôle détermine entre autre ce qu'un sujet doit faire dans l'activité pour en réaliser l'objet. Dans DARE, réaliser l'objet dans une certaine situation revient à réaliser la tâche. Il peut arriver qu'une division du travail identifie une sous-partie de l'objet pouvant être réalisé dans une activité externe. En fait, le sous-objet constitue lui-même un objet d'activité, mais dont le produit sera utilisé ailleurs.

Nous identifions un tel lien en terme de lien *sous-tâche*. La division du travail existe donc dans la tâche en terme de rôle et de sous-tâche. Les sous-tâches sont des tâches à part entière et peuvent elles-mêmes définir d'autres sous-tâches ou pré exister à la tâche qui en a besoin : une équipe dans un laboratoire se trouve souvent impliquée dans de nouveaux projets qui intègrent ainsi plusieurs activités complémentaires par rapport à l'objet du projet. De ce fait, une tâche peut être sous-tâche de plusieurs tâches. Les sujets qui réalisent une sous-tâche peuvent ou non être impliqués dans la sur-tâche. Un certain rôle dans une tâche peut avoir une influence sur le rôle d'un sujet aussi impliqué dans des sous-tâches : par exemple, le directeur du laboratoire tient une place particulière dans les activités de ses équipes de recherche.

La Figure 14 résume les relations qui viennent d'être décrites. Les sujets *a* et *b* sont impliqués dans une tâche *1* qui a pour sous-tâche la tâche *2*. Le sujet *a* n'est pas lui-même impliqué dans la réalisation de la tâche *2* dont le produit est quand même utile à la réalisation de la tâche *1*. Le sujet *b* participe à la réalisation des tâches *1* et *2*. Son rôle dans la tâche *1* a certainement une influence sur son rôle dans la tâche *2*. Le sujet *c* n'est pas directement impliqué dans la réalisation de la tâche *1*. Sa vision se limite à la tâche *2* qui sert pourtant à la tâche *1*. Celui-ci est aussi impliqué dans une tâche *3* qui ne possède pas de lien direct avec les tâche *1* et *2* : le seul lien entre ces tâches est le sujet *c*. N'oublions pas que, même si elles sont

conceptuellement déconnectées, la tâche *3* peut influencer la réalisation de la tâche *2* et donc de la tâche *1* par l'intermédiaire du sujet *c*. Cette relation est bien entendu bilatérale et la tâche *1* peut très indirectement influencer le travail du sujet *d* qui n'est pourtant lui-même impliqué que dans la tâche *3* !

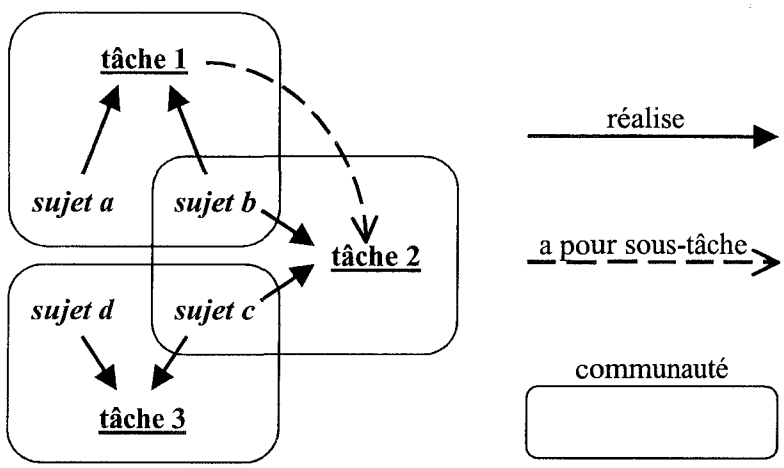

Figure 14. Définition de liens entre tâches.
Les sous-tâches sont des tâches à part entière.

Nous voyons en quelle mesure le lien tâche / sous-tâche permet de modéliser des organisations en utilisant l'activité, ou plutôt sa tâche, comme composant de base. De ce point de vue, une organisation est un ensemble de composants tâches reliés par des liens *sous-tâche*. Ce lien peut même être enrichi de manière à prendre en compte des relations de temporalité, définissant des relations séquentielles ou parallèles entre tâches. Une tâche est finie lorsqu'il existe un résultat produit correspondant à la représentation que les personnes se sont faite du but fixé. Le résultat est alors utilisable dans la ou les sur-tâches. Toutefois, les problèmes apparaissent lorsque les sujets créent par exemple des liens circulaires entre tâches. Le cas caricatural est celui où personne ne travaille plus et attend le résultat des autres. Il existe de nombreux problèmes de ce type proches de ceux du workflow, qui est un domaine de recherche à lui seul !

Dans DARE pour l'instant, nous avons délibérément choisi de laisser de côté ces problèmes de type workflow. Autrement dit, nous avons laissé de côté[1] les relations temporelles pouvant exister entre des tâches. Nous nous sommes plus précisément concentrés sur les fondements

[1] Nous reviendrons néanmoins sur ce sujet délicat dans le chapitre 8.

de l'architecture du système qui répondent à ceux de l'AT. L'AT nous explique que la division du travail, l'organisation elle-même, est construite par les sujets au fur et à mesure de leur activité. Il s'agit de l'expansivité de Kuutti appliquée au contexte organisationnel (cf. chapitre 3). Dans un premier temps, notre but par rapport aux liens entre tâches est de répondre à cette expansivité. Les sujets doivent pouvoir créer de nouvelles tâches et/ou créer des liens vers des tâches existantes au cours de leurs activités. Ceci revient à proposer la tailorabilité au niveau d'abstraction de la tâche. Relier des tâches existantes est de l'intégration. Créer de nouvelles tâches est de l'extension réalisée dans un langage de spécification de la tâche. Chaque tâche est elle-même un composant. Un composant tâche peut contenir des liens vers d'autres tâches (des sous-tâches), ces liens constituent une organisation, comme une université virtuelle par exemple. Ainsi, chaque composant tâche peut lui-même être composé d'une hiérarchie de composants tâche qui s'adressent à des niveaux d'abstraction différents et spécifiques d'une organisation. Dans une université virtuelle, les composants tâche peuvent correspondre aussi bien à l'administration centrale qu'à un module de travaux dirigés dans une matière particulière. Dans DARE, les tâches évoluent au cours de leur réalisation en étant manipulées par les sujets en activité. De ce fait, les composants tâche cristallisent l'héritage historique et culturel de la situation. Ainsi, *un composant tâche contient en lui l'expérience de toute une organisation, en terme d'outils, de rôles et de sous-tâches, pour un domaine spécifique.*

5.3 Précisions quant aux éléments de notre modèle conceptuel d'activité

Nous avons identifié les différents concepts qui viennent participer à un support d'activité dans DARE. Toutefois, nous n'avons pas encore défini la cardinalité de ces éléments. Comme nous allons le voir, certaines informations peuvent être retirées de l'AT alors que d'autres correspondent à des choix arbitraires qui, à notre avis, n'entament aucunement la validité de notre approche. Nous profiterons de ces précisions pour fournir quelques informations supplémentaires quant à notre vision des éléments du modèle pour la réalisation de DARE.

5.3.1 La tâche

Selon la définition de la tâche qui a été présentée, un support d'activité ne correspond qu'à une seule tâche. A l'inverse, nous verrons que les mécanismes mis en place dans DARE permettent de créer plusieurs supports d'activité qui correspondent à une même tâche. Ce type

d'utilisation du système peut par exemple s'avérer utile dans un contexte d'apprentissage où un enseignant veut impliquer des étudiants dans des activités séparées qui tentent de réaliser une même tâche fortement contrainte (ne permettant pas sa redéfinition). Ce type de déviation dans l'utilisation de DARE pourrait être exposé en terme d'appropriation dans le cadre de la théorie instrumentale (cf. chapitre 3), ses utilisateurs lui découvrant des schèmes d'utilisation pour lesquels il n'a pas réellement été conçu.

Dans DARE, la tâche est un composant qui sert de base à la création d'une organisation. Une organisation est elle-même une tâche construite par des sujets, en associant des tâches (en terme de sous-tâches), grâce à une tailorabilité hiérarchique, par intégration (si la sous-tâche préexiste), ou par extension (si la sous-tâche est créée pour l'occasion).

5.3.2 L'objet

L'élément qui est à l'origine d'une activité est l'objet. L'AT définit l'objet comme l'élément qui différencie une activité d'une autre. De par cette définition, il ne peut exister qu'un seul objet par activité. Dans DARE, l'objet est spécifié dans la tâche. Il n'est pas lui-même un composant, mais plutôt un attribut de la tâche. Il permet aux sujets de se reconnaître dans l'activité car selon la définition du TCAO donnée par Kuutti [69], les sujets sont actifs et de ce fait comprennent et sont en accord avec l'objet de l'activité.

5.3.3 Le sujet

Pour exister, l'activité doit au moins comporter un sujet. En effet, la tâche ne peut être réalisée s'il n'y a pas de sujet qui y travaille. Pour DARE, une activité n'impliquant aucun sujet correspond simplement à une tâche.

Le sujet[1] est une personne impliquée dans l'activité. Dans DARE, le sujet n'existe qu'au niveau du support d'activité où lui est affecté un certain rôle ayant été défini dans la tâche. Grâce à ce rôle, il devra et pourra utiliser une partie des outils informatiques qui ont eux aussi été spécifiés dans la tâche.

La nature du sujet est complexe. Il contient en lui les éléments de l'activité implicites qui n'ont pas été spécifiés dans la tâche. Ainsi, les actions du sujet sont non seulement

[1] Rappelons toutefois que dans le modèle d'Engeström, à la différence de notre approche, un sujet ne correspond pas forcément à un individu isolé.

influencées par sa représentation dans le support d'activité, mais aussi par un ensemble de contraintes (règles, division du travail, implication dans d'autres activités) implicites du point de vue de DARE, et donc extérieures à l'environnement médiatisé par le collecticiel. Comme nous l'avons déjà dit, DARE ne tente pas de modéliser l'humain mais lui offre un support informatique. C'est la raison principale pour laquelle un support d'activité dans DARE ne contient pas tous les éléments qui participent à l'activité coopérative, mais seulement une partie spécifiée et gérée par des sujets qui seuls ont une vision globale de la situation réelle.

5.3.4 La communauté

La communauté est définie comme l'ensemble des sujets qui partagent le même objet. De ce fait, il ne peut exister qu'une communauté par activité. Toutefois, n'oublions pas pour autant qu'un sujet dans une activité peut être impliqué en tant que sujet dans une autre activité. Nous avons déjà exposé qu'AT, et en particulier Kuutti, accorde une grande importance à la participation des personnes dans plusieurs activités parallèles et parfois contradictoires. Une personne peut appartenir à plusieurs communautés, ce qui pourra être la cause de conflits qui influencent le cours des activités auxquelles elle participe. Certaines de ces communautés peuvent utiliser DARE pour supporter leur activité, mais la plupart sont totalement extérieures à l'environnement médiatisé.

Dans DARE, la communauté, tout comme les sujets, n'existe qu'au niveau du support d'activité. Elle n'est représentée dans la tâche que par l'ensemble des rôles qui y sont spécifiés et qui pourront être joués par ses sujets.

5.3.5 L'outil

Le nombre d'outils peut varier d'une activité à une autre et les outils qui sont utilisés pour réaliser la tâche dépendent principalement de l'objet de l'activité et des conditions environnementales de l'activité réelle. En particulier, dans les activités distribuées qui nous intéressent, l'activité met toujours en jeu au moins un outil : le collecticiel ! Dans notre cas, la particularité de DARE est qu'il se considère lui-même comme un outil à part entière dans le support qu'il offre à l'activité coopérative. Nous reviendrons plus tard sur cette notion quelque peu déroutante (cf. chapitre 7). Retenons que, dans une activité supportée par DARE, le nombre d'outils n'est à priori pas limité et dépend des besoins des sujets.

Dans DARE, les outils sont vus comme des composants logiciels. Ils sont intégrés dans un support d'activité par le biais de la tâche correspondante. Ce sont en fait des applications informatiques coopératives ou non. De tels outils sont par exemple un tableau blanc partagé, un outil d'audio conférence, une calculette ou encore un éditeur coopératif. Nous verrons plus loin (cf. chapitre 7) que dans DARE les outils sont en partie réalisés par des applets Java. Comme nous l'avons précisé dans le chapitre 1, notre but n'est pas de créer des outils coopératifs particuliers, mais de fournir un environnement intégrateur. A priori, toute applet découverte sur Internet par une communauté peut devenir un outil dans un support d'activité. DARE joue le rôle d'environnement intégrateur dont la glu est l'activité coopérative. Les outils sont donc développés totalement en dehors de DARE. Nous ne pouvons pas connaître à l'avance les outils qui seront intégrés dans l'environnement de TCAO par les sujets au cours de leurs activités coopératives.

Une dernière remarque : les seuls outils développés dans le cadre de cette thèse sont ceux qui permettent de manipuler DARE. Ce sont cependant pour DARE des composants outils à part entière ! (cf. chapitre 7)

5.3.6 *Le rôle et les micro-rôles*

Le nombre de rôle n'est pas fixé. Pour DARE, il existe toujours au moins un rôle mis en jeu dans le support d'activité. Ceci correspond aux cas où tout le monde possède le même rôle. Ce cas arrive en particulier lorsque aucun rôle n'a été spécifié dans la tâche. Un même rôle peut donc être joué par plusieurs sujets. Il est plus difficile de déterminer si un sujet peut où non jouer plusieurs rôles au sein d'une même activité. La notion de rôle ayant été introduite pour synthétiser les règles et division du travail explicites, l'AT ne nous fournit pas d'information à ce sujet. Les règles et division du travail concernant un sujet au sein d'une activité peuvent être traduites en un seul ou plusieurs rôles. Il s'agit là d'un problème de conceptualisation et d'abstraction qui, même s'il est lié à nos travaux, n'en constitue pas l'objet principal : qu'ils possèdent un ou plusieurs rôles dans une activité, les sujets doivent être capables de le ou les faire évoluer en accord avec leurs besoins émergeants. Ce sont les moyens à mettre en œuvre pour supporter cette évolution qui nous intéressent. Toutefois, pour vérifier la faisabilité de notre approche en terme de modèle générique et d'architecture logicielle, il nous a fallu faire un choix. Dans DARE, chaque sujet ne possède qu'un rôle global par activité, donc un seul des rôles définis dans la tâche correspondante. Ce rôle est lui-

même constitué de plusieurs micro-rôles qui décrivent les droits et devoirs du sujet envers chaque outil spécifié dans la tâche.

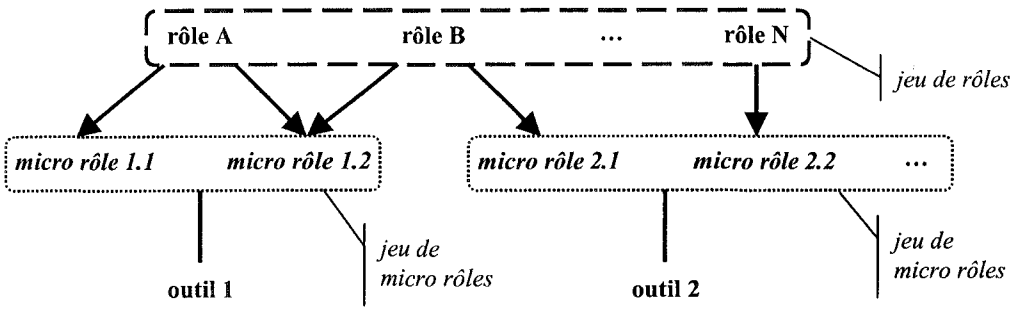

Figure 15. Les rôles, les micro-rôles et les outils.
Chaque micro-rôle définit des droits et des devoirs envers un outil. Un rôle est un agrégat de micro-rôles.

La Figure 15 représente une partie des éléments spécifiés dans une tâche. Une tâche définit un ensemble de rôles. Dans DARE, chaque rôle est un composant qui peut être réutilisé et/ou modifié. Les différents rôles sont généralement complémentaires en traduisant par exemple une certaine division du travail. C'est pourquoi cet ensemble de rôles est défini comme un jeu de rôles. Comme nous l'avons vu, la tâche définit aussi un ensemble d'outils qui permettent aux sujets de réaliser leur travail en fonction de leur rôle.

L'utilisation d'un outil particulier peut être vue comme une micro-activité (une sorte de sous-tâche définie dans l'outil). De ce fait, nous définissons un jeu de micro-rôles qui définit des droits et devoirs envers chaque outil. Chaque micro-rôle est un composant qu'il est possible de réutiliser en le liant à un autre outil. Par extension, il est aussi possible de réutiliser un jeu de micro-rôles.

5.3.7 Synthèse sur les éléments du modèle conceptuel

La Figure 16 synthétise les différents points que nous venons de présenter. En particulier, elle représente les principales interrelations existant entre les concepts définis dans la structure basique d'une activité d'Engeström et ceux que nous avons introduits pour concevoir notre modèle conceptuel d'activité.

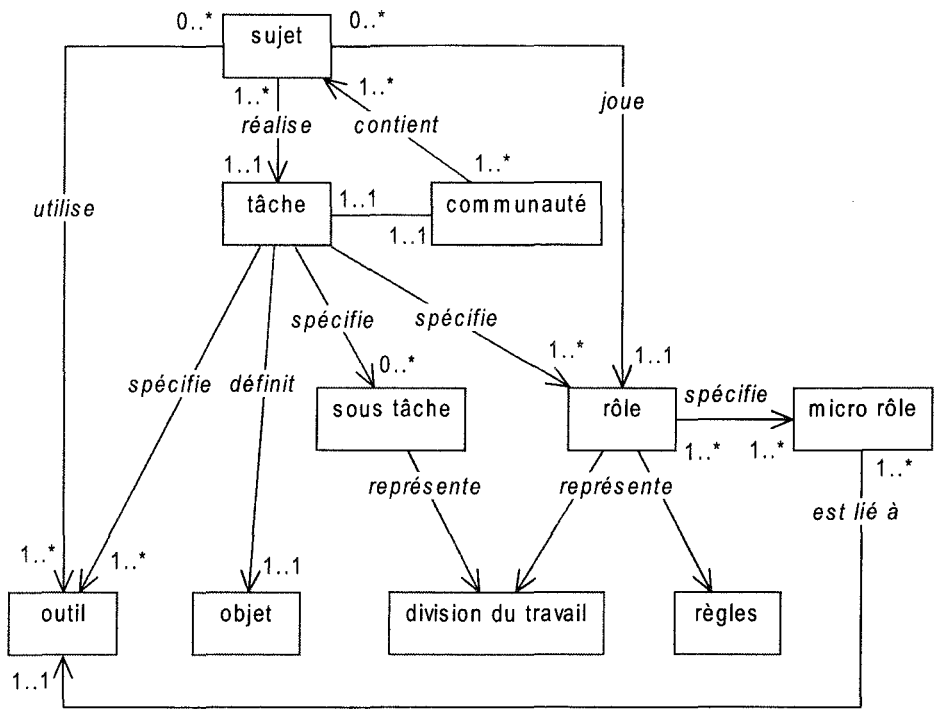

Figure 16. Les concepts de la structure basique d'une activité et ceux du modèle conceptuel de DARE

5.4 Le modèle conceptuel de DARE

Nous définissons le support d'activité comme une instance de tâche. Rappelons que le support d'activité est le support offert par DARE à l'activité, ainsi défini pour le différencier de l'activité réelle dans laquelle DARE est lui-même engagé.

La Figure 17 schématise les relations existantes entre les différents éléments qui participent au support d'activité dans DARE. Un support d'activité est construit à partir d'une tâche qui définit l'ensemble des outils, l'ensemble des rôles mis en jeu (chaque rôle est lui-même constitué d'un ou de plusieurs micro-rôles qui n'ont pas été représentés pour augmenter la lisibilité de la figure), et l'ensemble des sous-tâches de cette tâche. La tâche définit aussi l'objet de l'activité. Le sujet fait partie d'une communauté engagée dans l'activité. Ses actions sont influencées, en terme de droits et de devoirs, par le rôle qu'il possède.

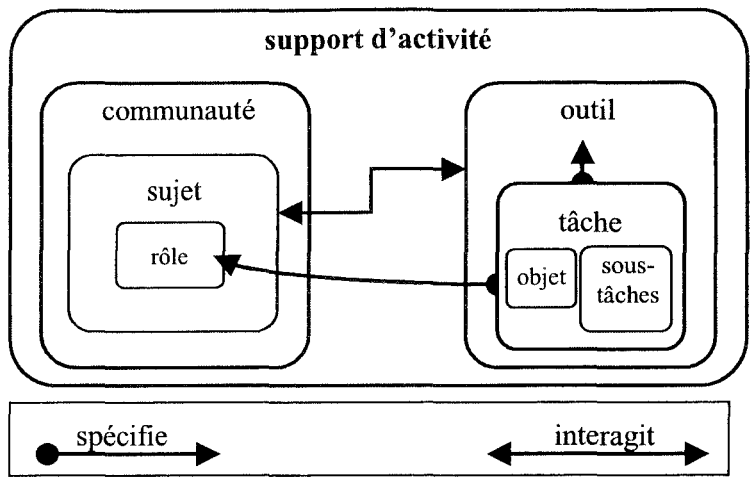

Figure 17. Le support d'activité et sa tâche : un système réflexif

Dans le support d'activité, le sujet participe et réalise l'activité en interagissant avec les outils qui ont été spécifiés dans la tâche. Il y a réellement interaction avec l'outil car, dans cet environnement médiatisé par l'ordinateur, toute action du sujet ainsi que tout retour d'information le concernant est réalisé au travers d'outils informatiques. Les outils peuvent être de natures très diverses. Il peut entre autre s'agir d'outil de communication (comme un outil d'audio conférence), d'outils permettant l'accès à des documents partagés (comme un tableau blanc ou un éditeur de texte coopératif), ou encore d'outils individuels (telle une calculatrice). Nous reviendrons dans les chapitres 6 et 7 sur ce que sont exactement les outils dans DARE. Notre intérêt est ici d'insister sur l'étude de la Figure 17 qui présente un point capital de notre conception : *la tâche est rendue accessible par le biais de certains outils du support d'activité.*

Pour atteindre nos objectifs, nous devons offrir aux sujets la possibilité de consulter ou de modifier la tâche du support d'activité dans lequel ils travaillent. Cet accès doit être possible au sein du support d'activité lui-même. Rappelons que la redéfinition de la tâche doit pouvoir être réalisée coopérativement. La fonction principale du support d'activité offert par DARE étant de créer une glu coopérative autour des outils qui y apparaissent, notre idée est de fournir un accès à la tâche au travers d'un ou de plusieurs outils particuliers. Puisque la tâche contient le modèle du support d'activité en cours d'exécution, les outils de manipulation de la tâche constituent une sorte de porte vers le noyau de DARE. Ces outils sont gérés dans DARE comme tous les autres outils. De cette manière, le support d'activité est en mesure de gérer l'accès à sa propre tâche selon les rôles qui y ont été définis. *Les fonctions de redéfinition du support d'activité sont ainsi totalement et uniformément intégrées dans l'activité coopérative.*

Notons que c'est la tâche qui spécifie les outils et les rôles mis en jeu dans le support d'activité. Ainsi, la tâche spécifie elle-même la liste des outils qui permettent de la modifier et le type de coopération mis en place pour le faire. Réciproquement, les outils de modification de la tâche permettent de la faire évoluer et donc de re-spécifier leur propre présence ou mode d'utilisation. Pour que ceci soit réalisable au cours de l'exécution du support d'activité, *nous devons mettre en place des mécanismes particuliers qui garantissent que toute modification de la tâche implique des conséquences directes sur le support d'activité.*

Pour conclure, retenons que dans le modèle conceptuel de DARE, la tâche est elle-même un artefact accédé et modifié par les sujets en fonction de leur rôle. Ainsi, *la tâche est modifiée au sein du support d'activité qu'elle décrit dans une relation causale, ce qui correspond à un système réflexif.* Cette propriété permet au sujet de modifier la situation informatisée qui influence son activité coopérative. Concrètement, ceci signifie que le sujet va pouvoir modifier l'ensemble des artefacts (outils et rôles) de son support d'activité. Pour atteindre nos objectifs, DARE, doit implémenter ce modèle conceptuel et répondre aux propriétés définies. Les chapitres 6 et 7 présentent cette implémentation.

5.5 Résumé sur le concept de tâche

Ce chapitre a présenté l'importance prise par le concept de tâche dans nos travaux. Ce concept a été identifié dans le but de fournir aux sujets un moyen d'accéder à la malléabilité de leur collecticiel. Grâce au concept de tâche, ce moyen est proche du niveau d'abstraction du support à l'activité qu'ils utilisent. La tâche est utilisée par les sujets pour spécifier leurs besoins émergeants. De cette manière, les sujets ne re-programment pas leur support d'activité, mais re-spécifient la tâche correspondante. Le Tableau 6 résume les points principaux que nous avons identifiés pour le concept de tâche.

- La tâche est le concept unificateur qui permet de décrire un support d'activité dans son propre contexte.
- La tâche spécifie l'objet de l'activité, les outils, les rôles et les sous-tâches.
- La tâche est un patron d'activité.
- La tâche est modifiée au sein du support d'activité qu'elle décrit dans une relation causale.
- La tâche est l'élément du plus haut niveau d'abstraction qui cristallise l'expérience des sujets d'une organisation toute entière.

Tableau 6. Le concept de tâche

Chapitre 6

L'implémentation ouverte

Ce chapitre présente les techniques de conception logicielle avancées et leur utilisation dans l'implémentation du noyau de notre réalisation. Cette implémentation tire directement partie des résultats exposés précédemment et son architecture repose en particulier sur le modèle conceptuel présenté dans le chapitre 5.

Pour commencer, nous reformulons le problème qui nous intéresse du point de vue de l'ingénierie logicielle. Nous présentons ensuite les techniques que nous jugeons utiles et particulièrement adaptées pour atteindre nos objectifs. En particulier, nous montrons comment nous avons lié l'approche de l'implémentation ouverte à celle de la mise en œuvre d'une architecture cadre (cf. chapitre 4) pour le TCAO. Cette démarche nous permet de créer un système réflexif permettant de supporter les activités coopératives et basé sur les concepts et mécanismes proposés par la Théorie de l'Activité (AT).

6.1 Reformulation du problème posé par la conception des environnements évolutifs pour le TCAO

D'une manière générale, les systèmes de TCAO réalisent une tâche qui a été construite grâce à une analyse réalisée dans des conditions spécifiques. Malheureusement, des problèmes d'appropriation apparaissent souvent, et parfois même, le système est rejeté, ceci étant généralement dû à un décalage existant entre la tâche du système et celle de l'utilisateur (ou la représentation qu'il s'en est construite). Si l'utilisateur se trouve dans l'incapacité de comprendre la tâche que le système implémente réellement, ou d'adapter cette tâche à la sienne, le système ne sera certainement pas utilisé.

Dans notre modèle, nous avons défini que la tâche contient une représentation du support d'activité. Notre idée est de permettre aux sujets d'accéder à cette représentation dans le but de les aider à comprendre le système et, si nécessaire, à le modifier. Cette approche correspond à celle de l'implémentation ouverte (open implementation) décrite par Kiczales [63] dans le domaine de la conception logicielle.

6.2 L'implémentation ouverte

L'implémentation ouverte discute des limites de l'abstraction de la *boîte noire* largement utilisée dans l'ingénierie. Nous avons déjà introduit cette notion dans le chapitre 4 en discutant des architectures cadres boîte noire et boîte blanche.

Comme le souligne Kiczales [63], le problème fondamental de l'ingénierie est de contrôler la complexité. Les systèmes construits sont si complexes qu'il est impossible d'en comprendre l'ensemble en une fois. C'est pourquoi la décomposition et l'abstraction sont indispensables à la modélisation. Rumbaugh définit l'abstraction comme « un moyen d'isoler les aspects importants et de supprimer ceux qui ne le sont pas » [115]. L'abstraction est largement utilisée dans l'ingénierie logicielle et les méthodes de modélisations les plus développées comme OMT ou UML reposent sur celle-ci. Un modèle est en effet une abstraction de quelque chose qui permet de comprendre avant de construire.

L'abstraction la plus répandue dans l'ingénierie est celle de la *boîte noire* qui permet de créer un système à partir de modules fonctionnels. Chaque module est explicitement séparé en une interface fonctionnelle et une implémentation. L'interface fonctionnelle est ce qui est proposé

au client pour l'utilisation du module, alors que l'implémentation est encapsulée. Ce principe facilite la réutilisation des modules créés en présentant des propriétés similaires aux composants. Les composants logiciels (cf. chapitre 4) sont d'ailleurs généralement implémentés sous forme de boîtes noires.

Toutefois, Kiczales explique que la boîte noire doit parfois être ouverte. En effet, celui-ci souligne que la stratégie d'implémentation d'un module ne peut parfois être déterminée que par celui qui sait comment ce module va être utilisé. Le but est donc de permettre aux utilisateurs du module de comprendre les stratégies d'implémentation sous-jacente et/ou de leur permettre de choisir une stratégie différente qui réponde mieux à leurs besoins. Cependant, ceci ne doit pas totalement remettre en cause le principe de la boîte noire. L'approche adoptée par l'implémentation ouverte peut alors se résumer ainsi :

- Un module doit être utilisable sans que son utilisateur ait à se soucier de son implémentation : ceci permet de conserver les propriétés intéressantes de l'approche boîte noire lorsque l'implémentation est adéquate.
- La stratégie d'implémentation d'un module doit pouvoir être contrôlée lorsque le besoin s'en fait sentir.
- Ces deux types de manipulation d'un module doivent être accessibles mais indépendants.

De ce fait, l'implémentation ouverte d'un module présente deux interfaces : La première interface, dite *interface de base*, permet d'accéder aux fonctions du module. La seconde interface, dite *interface méta*, permet à l'utilisateur d'ajuster la stratégie d'implémentation existante sous l'interface de base (cf. Figure 18). Le problème est alors d'identifier un moyen pour fournir une telle interface méta. Une solution réside dans la mise en œuvre d'un système réflexif.

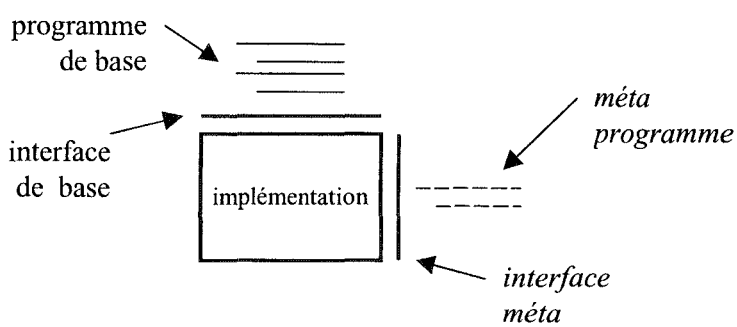

Figure 18. L'implementation ouverte (repris et adapté de [63])

6.3 Les systèmes réflexifs

"A reflective system is a causally connected meta-system that has as object system itself " ([78], p 19)

Maes [78] définit un *système informatique* comme des données, un programme et une exécution qui représentent une partie du monde réel. Ce qui est représenté est appelé *domaine* du système. Il existe une *relation causale* entre le système et son domaine si les modifications du domaine entraînent automatiquement des modifications de la représentation contenue dans le système. De la même manière, des modifications dans le système doivent automatiquement avoir des répercussions sur les parties correspondantes du domaine. Maes propose l'exemple d'un bras robot dans une chaîne de montage. Il existe généralement une relation causale entre les données qui représentent l'automate dans le système informatique et la position du bras dans le monde réel.

Un *méta-système*[1] est un système informatique qui a pour domaine un autre système informatique appelé *système objet*. De ce fait, le programme d'un méta-système est appelé un *méta-programme* qui raisonne et agit sur les entités et relations qui existent dans le système objet. Programmer un méta-système correspond à de la *méta-programmation*. Il peut bien entendu exister une relation causale entre un méta-système et son système objet.

Un *système réflexif* est un méta-système ayant une relation causale avec un système objet qui est lui-même. Ainsi, un système réflexif est un système qui contient à la fois des représentations d'une partie du monde réel, et des représentations de sa propre activité. Un système réflexif est donc capable d'examiner son propre état et sa propre structure. La relation causale implique que toute modification de la représentation méta est automatiquement répercutée vers le comportement du système. L'utilisation des propriétés réflexives peut être séparée en deux fonctions principales : la première correspond à simplement examiner le système sans le modifier : il s'agit de l'*introspection* ; la seconde correspond à modifier le système et donc redéfinir sa structure et/ou son comportement : il s'agit de l'*intercession*.

Les systèmes réflexifs possèdent intrinsèquement deux facettes : une facette qui correspond au domaine adressé par le système, et une facette méta qui permet d'accéder à

l'implémentation du système lui-même. Il est alors aisé d'envisager en quelle mesure la réflexivité permet de mettre en œuvre l'implémentation ouverte. Chaque module peut être construit comme un module réflexif. Il offre d'une part une interface de base permettant d'accéder à sa représentation du monde réel. D'autre part, il fournit une interface méta correspondant aux fonctionnalités réflexives du module qui permettent d'accéder en introspection ou en intercession aux stratégies d'implémentation sous-jacentes. De ce fait, l'implémentation ouverte offre à la réflexivité un moyen bien défini d'accéder à l'implémentation du système : une interface méta permet de créer une abstraction méta adaptée au module pour accéder au niveau méta.

La réflexivité et l'implémentation ouverte ont déjà largement été explorées dans le domaine des langages de programmation. Elles ont été utilisées pour fournir les moyens qui permettent de fusionner l'environnement d'exécution des programmes et le langage lui-même. En particulier, une des applications les plus répandues de la réflexivité est la création de dévermineurs. Une autre utilisation est celle qui consiste à créer des langages capables de se redéfinir eux-mêmes, comme Common Lisp Object System (CLOS) [103] ou Smalltalk. Nous allons maintenant présenter comment l'implémentation ouverte et la réflexivité permettent de répondre à nos problèmes.

6.4 La malléabilité et l'implémentation ouverte

Une application informatique (comme un collecticiel) peut être considérée comme un module. Elle est elle-même un système informatique qui représente un domaine particulier. Classiquement, une application correspond bien à une boîte noire dont l'implémentation est inaccessible aux utilisateurs finaux. L'accès à l'implémentation est réalisé grâce à une méta-application qui, dans la plupart des cas, correspond à l'environnement de programmation dans lequel l'application est développée.

Rappelons que notre but est de mettre en place un système malléable (cf. chapitre 1 et 5). La malléabilité que nous avons présentée est offerte à l'utilisateur final par un accès à l'implémentation de l'application. On peut donc réaliser une application malléable par une application réflexive dont l'interface méta est rendue accessible à l'utilisateur final. En utilisant la malléabilité d'une application, l'utilisateur ne focalise plus directement sur le domaine adressé mais sur l'implémentation de son application et, de ce fait, passe d'une

[1] Ce concept, pour le TCAO, est en rapport avec celui de méta-collecticiel que nous avons introduit dans le chapitre 1.

interface à l'autre. Néanmoins, ces deux interfaces étant accessibles au sein de l'application, le passage de l'une à l'autre peut être réalisé sans rupture de contexte.

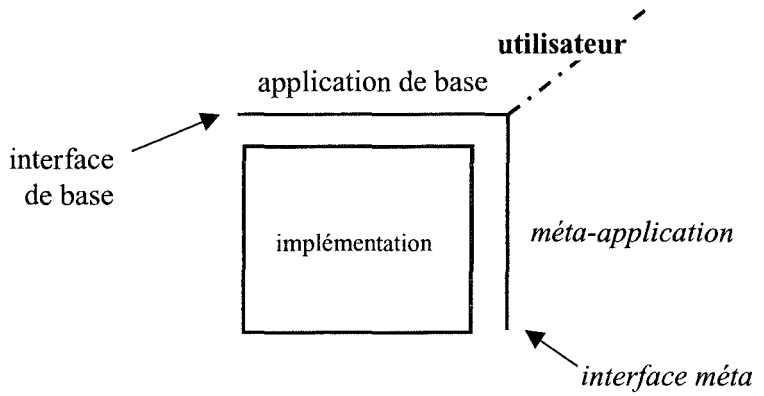

Figure 19. Une application malléable basée sur l'implémentation ouverte.
L'utilisateur est à la fois impliqué dans l'application de base et la méta-application

La Figure 19 représente une application malléable basée sur l'implémentation ouverte. Celle-ci fournit une interface de base qui permet d'utiliser l'application telle qu'elle est, ainsi qu'une interface méta accessible dans une méta-application. Dans le cas d'une application malléable, l'utilisateur peut passer du niveau applicatif à son niveau méta, et vice versa.

Le concepteur d'une telle application se trouve alors face à un problème non trivial : la définition de l'interface méta. Nous retrouvons les problèmes posés par la malléabilité. L'implémentation ouverte, par l'adjonction d'une interface méta, permet de définir des abstractions sur l'implémentation. Le problème est de déterminer les niveaux d'abstractions offerts, ainsi que les mécanismes qui y sont associés. Notons de plus que ces éléments dépendent aussi de l'implémentation elle-même. Par exemple, dans notre approche, l'application est réalisée sur une architecture cadre dédiée à un domaine spécifique. Dans ce cas, l'interface méta peut directement reposer sur la structure sous-jacente qui définit les points chauds, les niveaux d'abstractions et les mécanismes accessibles.

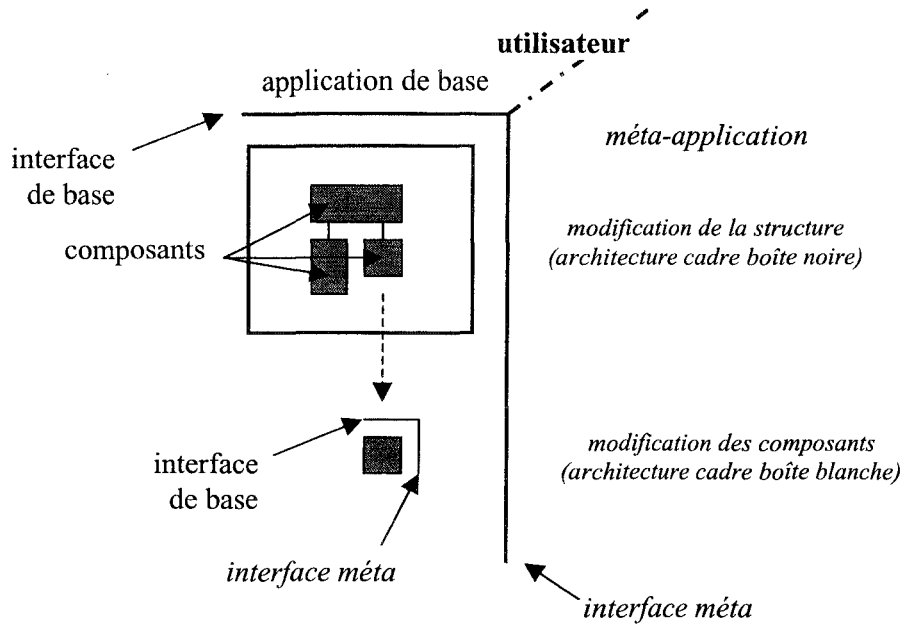

Figure 20. L'implémentation ouverte, et une application basée sur une architecture cadre

Pour réaliser DARE nous proposons de mettre en œuvre l'implémentation ouverte sur une architecture cadre. Cette proposition est représentée Figure 20. De cette manière, l'interface méta permet d'accéder à l'architecture cadre d'implémentation. Si cette architecture cadre est une boîte noire, l'interface méta permet de réorganiser les composants. Dans le cas où l'architecture cadre est aussi une boîte blanche, chaque composant est lui-même un module mettant en œuvre l'implémentation ouverte et fournissant de ce fait une interface méta permettant d'accéder à son implémentation. Ainsi, l'interface méta de plus haut niveau dépend de la structure de l'architecture cadre et des interfaces méta de ses composants.

Dans une approche hiérarchique, chaque composant est lui-même composé de composants. Ceci nous laisse imaginer une structure récursive de composants basés sur l'implémentation ouverte et offrant chacun une interface de base pour l'utilisation 'normale' et une interface méta pour la malléabilité.

6.5 L'implémentation ouverte dans DARE

Notre démarche est de lier l'approche adoptée par l'implémentation ouverte avec notre modèle conceptuel basé sur la Théorie de l'Activité (AT). Il existe aujourd'hui plusieurs langages de programmation qui permettent de supporter cette approche. En liant notre modèle

à l'implémentation ouverte, nous pensons pouvoir réutiliser les mécanismes réflexifs de ces langages pour implémenter DARE comme un système réflexif basé sur l'AT.

Dans notre approche des collecticiels, la boîte noire est le système de TCAO et, au niveau d'abstraction qui nous intéresse, la stratégie d'implémentation est la tâche sous-jacente. Nous avons déjà amplement exposé le fait que cette boîte noire doive parfois être ouverte par les sujets en cours d'activité de manière à en comprendre et/ou adapter la tâche.

Cette démarche nous permet d'adapter l'approche de l'implémentation ouverte à l'architecture cadre définie par l'AT. Les sujets interagissent grâce à un support d'activité qui réalise une tâche spécifique. Du point de vue du sujet, le support d'activité est lié à la tâche par une interface de base qui correspond à l'utilisation 'normale' du collecticiel. En cas de problème, c'est-à-dire lorsqu'un conflit éclate dans l'activité, les sujets passent dans une méta-activité qui correspond au niveau expansif de Kuutti et aux niveaux coopératifs et co-constructif de Bardram (cf. chapitre 3). Du point de vue de DARE, cette méta-activité correspond à une modification de la tâche et se déroule dans un support de méta-activité qui offre une interface méta pour modifier la tâche sur laquelle repose le support d'activité.

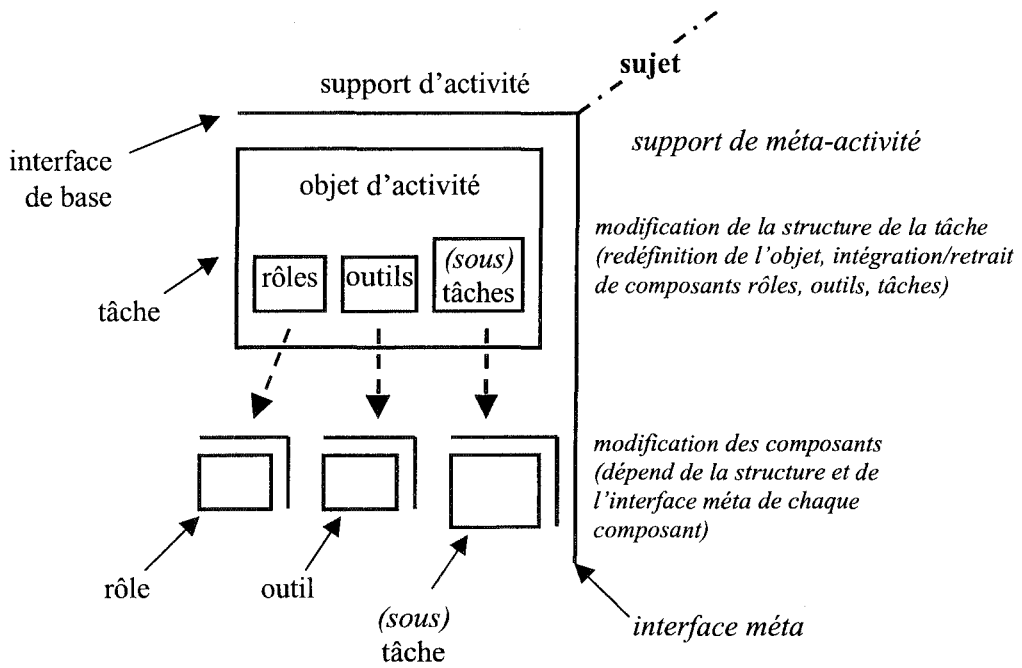

Figure 21. DARE : un collecticiel malléable basé sur le concept de tâche.

La Figure 21 représente l'approche de l'implémentation ouverte basée sur L'AT telle qu'elle apparaît dans DARE. Concrètement, un support d'activité est un outil qui assiste les sujets dans la réalisation d'une tâche en leur fournissant un environnement dans lequel apparaissent d'autres outils, des rôles et des sous-tâches. Certains des outils, appelés *outils méta*, servent au support de la méta-activité en offrant aux sujets une interface méta sur la tâche. Un exemple d'outil méta est celui qui permet de (re)définir un composant rôle. En particulier, l'outil méta de (re)définition d'une tâche utilise l'interface de modification de la tâche et permet d'ajouter et/ou de retirer des outils méta de la tâche, rendant ainsi le collecticiel plus ou moins malléable. Il est aussi possible de définir des règles et une division du travail pour la méta-activité. Pour ce faire, il suffit d'utiliser les outils méta pour définir des rôles sur eux-mêmes.

Notons que les composants apparaissant dans la tâche (rôles, outils, sous-tâche) sont eux-mêmes réalisés selon l'implémentation ouverte et fournissent de ce fait aussi chacun une interface de base (utilisation) et une interface méta (introspection et/ou intercession). En particulier, rappelons que nous définissons chaque rôle comme un ensemble de micro-rôles liés à des outils. Dans notre approche, l'interface méta de chaque rôle permet de modifier l'ensemble de ses micro-rôles. Chaque micro-rôle est de plus lui-même un composant réalisé selon l'implémentation ouverte, ceci permettant d'obtenir la malléabilité maximum.

6.6 Le Meta-Object Protocol

Le *Meta-Object Protocol (MOP)* [64] est une réalisation orientée objet (au sens de l'ingénierie logicielle) de l'implémentation ouverte et donc de la réflexivité. Le premier système qui ait implémenté le MOP est le langage Common Lisp Object System (CLOS) dont nous avons déjà parlé précédemment. Aujourd'hui, d'autres langages implémentent le MOP. On peut par exemple citer Smalltalk ou encore Java. Notre but n'est pas de présenter ici l'ensemble des motivations qui ont amené ses auteurs à créer le MOP, ni même d'en expliciter tous les fondements. Une large description peut être trouvée dans [64] et [103]. Notre objectif est de présenter les éléments nécessaires à la compréhension de l'implémentation de l'environnement DARE.

Dans les langages objets, le niveau d'exécution correspond aux objets d'implémentation. Chaque objet a une description qui correspond à une classe. De cette manière et comme le définit Rumbaugh, « une classe d'objets décrit un groupe d'objets ayant des propriétés similaires (attributs), un comportement commun (opérations), des relations communes avec d'autres objets ainsi qu'une même sémantique » ([115], p23).

Dans l'approche MOP, les classes sont elles-mêmes représentées par des objets qui sont décrits dans une méta-classe. La méta-classe est la classe d'une classe. Elle définit par exemple ce qu'est l'instanciation, le lookup, l'ajout de méthodes..., ce qu'il est possible de demander à toute classe dont elle est la classe. De ce fait, c'est dans la méta-classe que sont définis les mécanismes d'introspection et/ou d'intercession. Rappelons que l'introspection, dans les langages objet, permet d'examiner la description d'un objet, c'est-à-dire sa classe. L'intercession quant à elle permet de la modifier.

Dans Java, par exemple, la classe java.lang.Class est la classe de toutes les classes (existantes ou créées par les développeurs). java.lang.Class définit un ensemble de méthodes exécutables sur toute classe Java. Par exemple, la méthode java.lang.String getName() permet à toute classe de renvoyer son nom sous la forme d'une chaîne de caractères. En Java, tout objet hérite de java.lang.Object qui définit la méthode java.lang.Class getClass(). Ainsi, un objet en cours d'exécution est capable de récupérer sa classe et de lui demander quel est son nom. java.lang.Class définit aussi la méthode java.lang.reflect.Method[] getMethods() qui permet de récupérer l'ensemble des méthodes publiques d'une classe. C'est de l'introspection : la classe est examinée et non pas modifiée. Dans un environnement distribué où les objets peuvent migrer d'un environnement d'exécution à un autre, ce type de mécanisme s'avère très intéressant en permettant à un programme de 'découvrir' les capacités d'un objet inconnu. Ce sont aussi des mécanismes similaires qui permettent de réaliser des approches composants comme les JavaBeans [57].

Ainsi, dans le MOP, l'encapsulation objet de l'implémentation ouverte est réalisée par l'introduction de la méta-classe. En fait, la méta-classe définit dans un formalisme objet l'interface méta qui permet d'accéder à la description de l'implémentation, c'est-à-dire à la classe. Grâce à cette approche objet, le MOP ouvre de nouvelles perspectives sur la méta-programmation [103]. Il est par exemple tout à fait possible d'hériter d'une méta-classe pour en créer une nouvelle plus spécialisée. Celle-ci hérite des méthodes d'instance définies par la méta-classe originelle, et en définit des spécifiques. Toutes ces méthodes sont alors applicables aux classes instances de cette nouvelle méta-classe.

Dans DARE, nous avons utilisé les propriétés du MOP et des langages qui l'implémentent pour proposer un méta-modèle non plus en termes de classe, d'attribut ou encore de méthode, mais dans les termes de notre modèle conceptuel basé sur l'AT.

6.7 Modèle et méta-modèle

6.7.1 Un nouveau méta-modèle

Un méta-modèle est un modèle de modèle. Les informaticiens sont habitués à utiliser différents méta-modèles [45]. Par exemple, pour écrire un programme, le développeur doit maîtriser tout ou partie du méta-modèle de son langage de programmation. Ce langage, une fois approprié, lui permet d'implémenter de nombreux modèles d'implémentation sous forme de programmes. Un autre méta-modèle aujourd'hui bien répandu dans la communauté informatique est UML (Unified Modelling Language) [92].

Les éléments qui constituent un méta-modèle sont appelés *méta-types*. Par exemple, les méta-types d'UML (proches des méta-types des langages objet en général) sont la Classe, l'Opération, l'Attribut, ou encore l'Association. Les instances des méta-types permettent de créer des modèles. Un modèle UML sera donc fait de classes, d'attributs, d'opérations et d'associations, qui sont respectivement des instances des méta-types cités ci-dessus. Dans un système réflexif, les méta-types sont eux-mêmes implémentés et fournissent une interface méta. Dans l'approche du MOP par exemple, le méta-type Classe est implémenté sous forme de méta-classe (en Java, la classe correspondante est java.lang.Class) qui fournit une interface méta permettant, entre autres, de lister les méthodes de toute classe, donc de toute instance du méta-type.

La malléabilité telle que nous l'envisageons fournit un accès à l'utilisateur final à l'interface méta, de manière à ce qu'il puisse examiner ou modifier le modèle d'exécution de son application. Même si DARE est lui-même implémenté à partir du méta-modèle d'un langage objet, notre idée est de définir un nouveau méta-modèle mettant en jeu des méta-types d'un niveau d'abstraction plus proche de l'utilisateur final, c'est-à-dire d'un niveau plus élevé que celui des fondements du langage sous-jacent. Ainsi, à l'instar du méta-modèle d'un système objet qui définit le méta-type *Classe* dont l'instanciation permet de créer les représentations des objets que sont les classes, DARE définit le méta-type *Tâche* dont l'instanciation permet de créer des représentations des activités que sont les tâches.

méta modélisation	méta-type Classe	méta-type Tâche
modélisation	classe	tâche
	système objets	*DARE*

Figure 22. L'introduction d'un nouveau méta-modèle pour DARE

De la même manière qu'un système objet introduit d'autres méta-types que Classe (comme le méta-type Attribut d'UML par exemple), nous avons vu précédemment qu'une tâche est constituée d'autres composants qui sont les sous-tâches, les outils et les rôles. Les sous-tâches sont des tâches à part entière, elles sont de ce fait elles-mêmes instances du méta-type Tâche. Les méta-types qui décrivent les outils et les rôles sont respectivement le méta-type Outil et le méta-type Rôle. Chaque rôle étant lui-même constitué d'un ensemble de composants micro-rôles, chaque micro-rôle est lui-même défini par le méta-type Micro-rôle. La Figure 23 synthétise ces différents concepts.

méta modélisation	méta-type Tâche	méta-type Outil	méta-type Rôle	méta-type Micro-rôle
modélisation	une tâche	un outil	un rôle	un micro-rôle

Figure 23. Les éléments du méta-modèle DARE et leurs instances dans des modèles DARE

6.8 L'implémentation

6.8.1 Mise en œuvre du Meta-Object Protocol

Pour être implémenté, DARE repose bien entendu sur un langage de programmation général. De manière à pouvoir implémenter simplement nos méta-types, nous avons utilisé un langage orienté objet qui implémente le MOP. Ainsi, chaque élément du méta-modèle (méta-type) correspond à une méta-classe. Pour ce faire, ces nouvelles méta-classes héritent et spécialisent la méta-classe standard du langage utilisé. Chaque nouvelle méta-classe correspond à l'implémentation d'un méta-type de notre nouveau méta-modèle. Les éléments d'un modèle, c'est-à-dire les classes, sont des instances de leur méta-classe respective.

La Figure 24 synthétise notre démarche dans une notation UML. Nos méta-classes spécialisent la méta-classe Classe. Au niveau des classes, la Figure 24 utilise des noms génériques pour décrire les instances de chaque méta-classe. Par exemple, la classe UnOutil représente un outil spécifique comme un tableau blanc partagé. En effet, chaque méta-classe permet de créer un nombre quelconque de classes en fonction des besoins des sujets. Les instances de ces classes sont des objets qui participent au support d'activité.

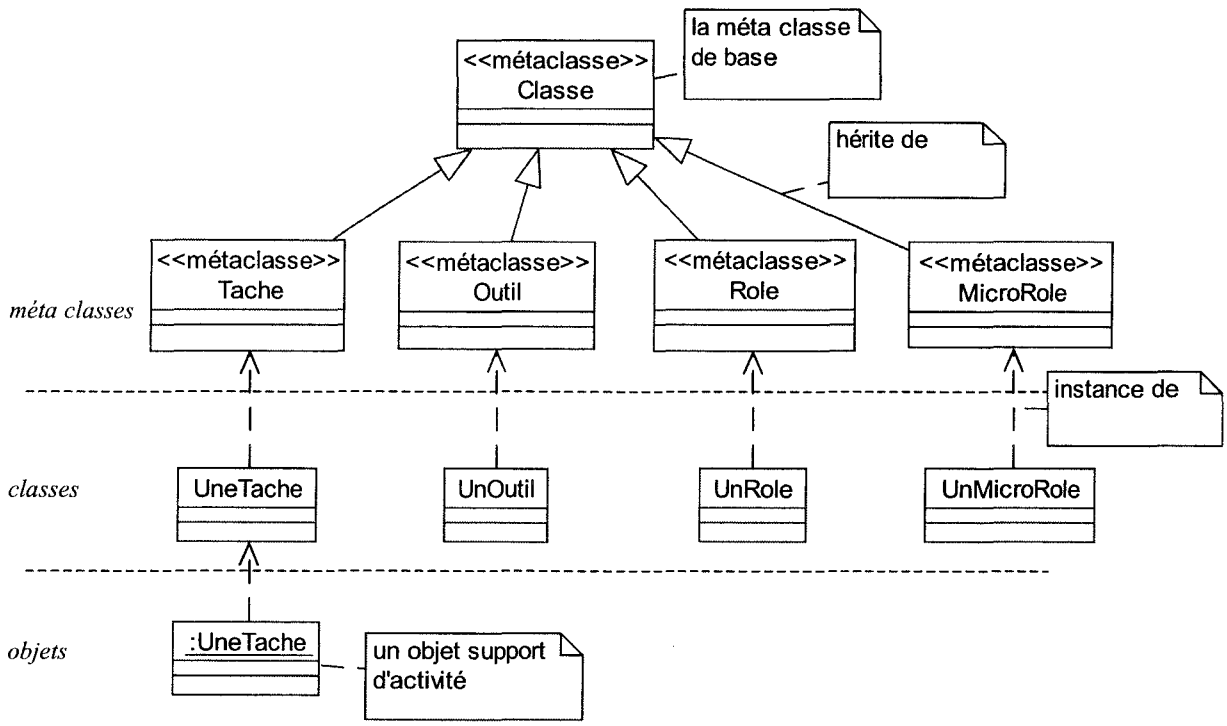

Figure 24. L'implémentation MOP du modèle conceptuel de DARE

6.8.2 Le méta-modèle

Il nous faut maintenant présenter ces méta-classes et leurs interrelations dans le méta-modèle, c'est-à-dire les liens qui existeront entre leurs instances. La Figure 25 représente ces interrelations dans une notation UML.

Une tâche définit et gère une collection d'outils. Ceci signifie implicitement que Tache contient les méthodes ajouterOutil(Outil), retirerOutil(Outil) ou encore listerOutils() qui permettent d'agir proprement sur l'ensemble des outils inclus dans une classe représentant une tâche. Il en est de même avec les collections de rôles et de sous-tâches. L'attribut nom permet de

demander à une tâche (instance de Tache) quel est son nom, ou encore de le modifier. Nous avons vu que l'objet (au sens de l'AT) d'une activité est défini dans sa tâche. De plus, comme les rôles ou les outils, l'AT explique que l'objet est lui-même susceptible de changer. C'est pourquoi, l'existence de l'attribut objet est définie dans la méta-classe Tache.

Dans notre approche, les utilisateurs doivent pouvoir définir ou redéfinir leurs outils. Nous verrons en détail dans le chapitre suivant que l'implémentation des outils (interface utilisateur et fonctions) est réalisée par des composants sous forme d'applets Java (utilisables dans un simple navigateur Internet). Ainsi, les instances des instances de Outil, contiennent en réalité une référence vers une applet Java instanciée dans le navigateur Internet d'un sujet. Leur principale fonction est donc de 'piloter' une applet sur un poste client. La méta-classe Outil spécifie de ce fait l'interface permettant de créer un lien entre une classe représentant un type d'outil et une applet Java. Ceci explique l'existence de l'attribut url. Cette URL (Uniform Resource Locator) contenue dans une classe représentant un outil particulier sera utilisée dans un support d'activité pour instancier l'applet Java référencée. La classe Operation représente les opérations qu'un outil peut exécuter grâce à la méthode executer(op:Operation). Par exemple, dans un outil TableauBlanc (une instance de Outil), les sujets pourront définir une opération effacer() qui, si elle est exécutée, déclenchera les méthodes adéquates sur l'applet. Les opérations peuvent être héritées d'un autre outil. Elle sont définies par les sujets (cf. Chapitre 7).

Un rôle définit et gère une collection de micro-rôles. Cette collection est construite par les sujets. MicroRole est liée à Outil et définit une collection d'actions pouvant être exécutées par la méthodes executer(a:Action). Les actions peuvent être héritées d'un autre micro-rôle et sont (re)définies par les sujets qui créent ou modifient un micro-rôle. Une action peut être visible ou non, ainsi que globale ou locale. L'utilisation de ces attributs sera détaillée dans le chapitre 7. Une particularité des actions est de demander à l'outil lié au micro-rôle d'exécuter certaines opérations. Ainsi, un micro-rôle contient un ensemble d'actions pouvant être déclenchées par un sujet qui le joue. Chaque action se traduit en un script d'opérations exécutées sur l'outil associé.

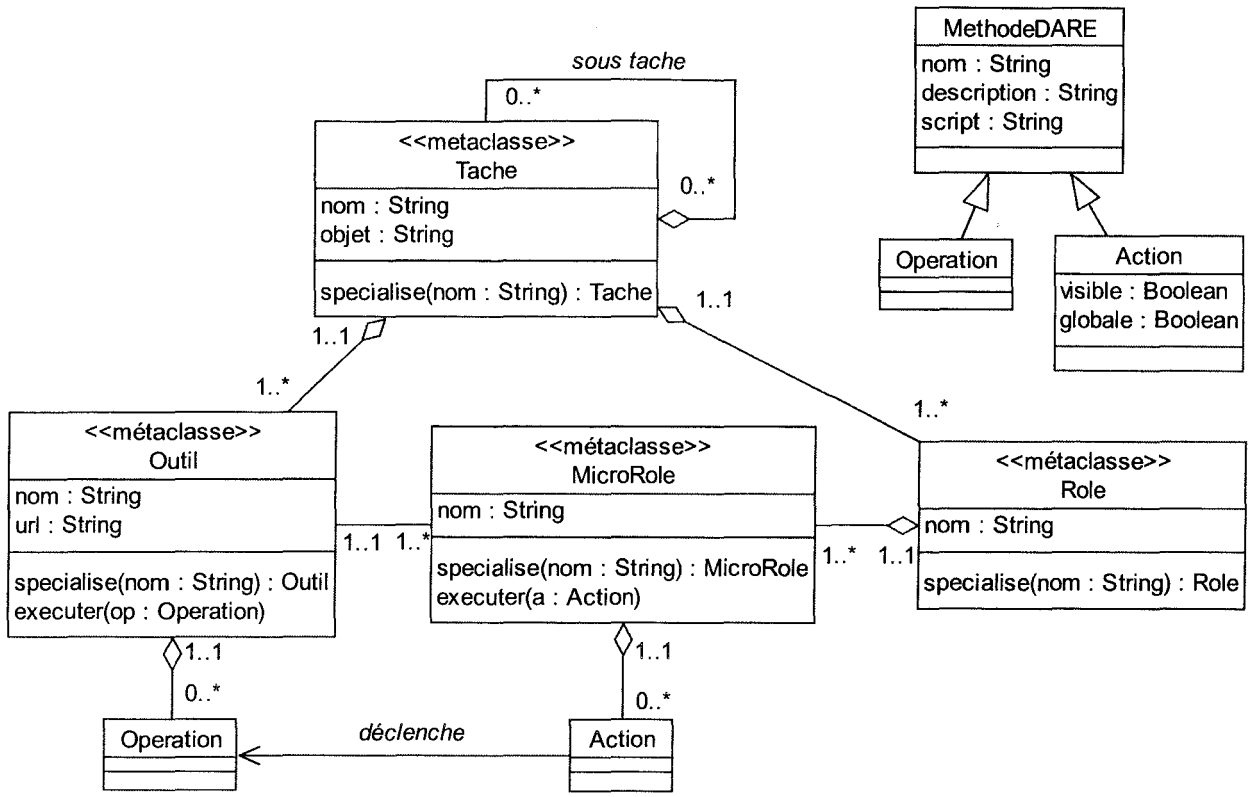

Figure 25. Le méta-modèle de DARE
NB : cette figure ne tient pas compte de l'approche composant réellement adoptée dans DARE. En effet, toute instance de ces méta-types peut exister sous forme de composant non connecté.

Enfin, il est possible d'hériter des classes instances de Tache, Outil, Role et MicroRole, ce qui permet aux tâches, outils, rôles et micro-rôles d'être construits à partir d'un élément existant et du même type. Pour ce faire, on utilise la méthode specialise de chaque méta-classe. Par exemple, l'appel de specialise sur une tâche UneTache avec comme paramètre *'NouvelleTache'* a pour effet de créer la tâche NouvelleTache qui hérite de UneTache. Il est ainsi possible de créer de nouvelles tâches à partir d'autres. Il en va de même pour les outils, les rôles et les micro-rôles. Les méta-classes ici définies permettent aux sujets de construire dynamiquement les classes des objets qui composent leurs supports d'activité. Ainsi, chaque classe représente un composant utilisable pour (re)définir une tâche et cristallise en elle une partie de l'héritage historique et culturel du support d'activité auquel ses instances participent. De ce fait, *la notion d'héritage est importante : elle permet de réutiliser l'expérience des sujets ayant cristallisé dans les classes.*

6.8.3 Un support d'activité

La Figure 26 est le modèle d'un support d'activité dans DARE. Les classes SupportActivite, OutilGen, RoleGen et MicroRoleGen sont des instances respectives de Tache, Outil, Role et MicroRole. Elles expriment d'une manière générale les liens entre les types d'éléments de DARE. La classe Membre représente un sujet dans un support d'activité. Chaque membre correspond à un utilisateur de DARE (Utilisateur). A l'inverse, un utilisateur de DARE peut être membre de plusieurs supports d'activités, mais pas plusieurs membres dans un même support ! Le membre joue un rôle et possède de ce fait un ensemble de micro-rôles qui lui donnent accès à un ensemble d'outils. Plusieurs instances de micro-rôles peuvent référencer la même instance d'outil. Les instances d'outils sont détenues par le membre et contiennent chacune une référence sur une applet. Chaque outil peut exécuter un ensemble d'opérations définies par les sujets. Cet ensemble n'est pas figé car de nouvelles opérations peuvent être définies dynamiquement par les sujets au cours de l'exécution du support d'activité, ceci grâce aux méthodes de la classe Outil présentées précédemment. Il en est de même pour l'ensemble des actions d'un micro-rôle.

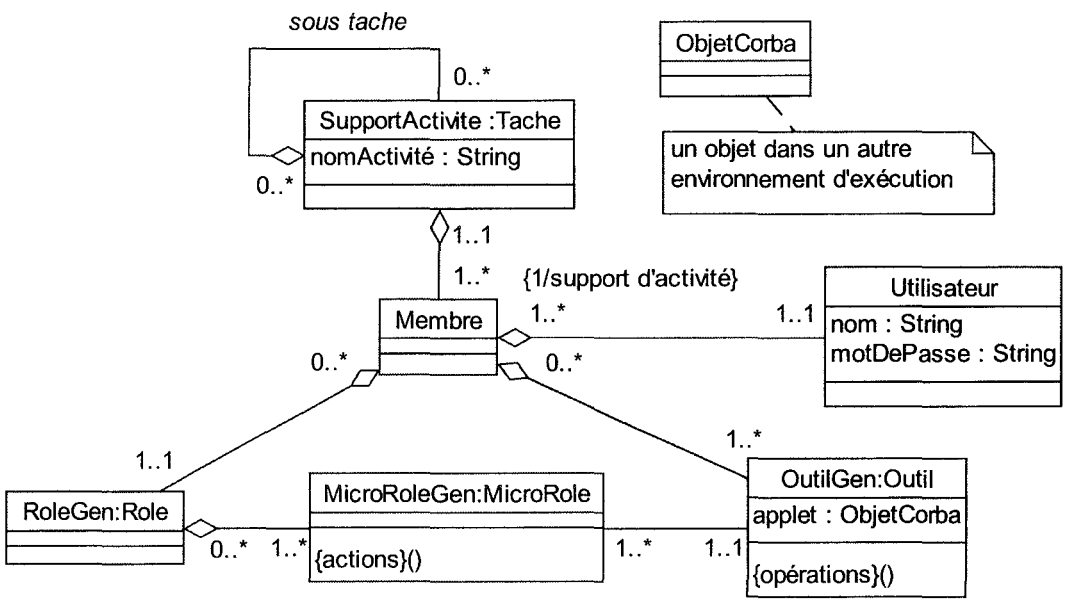

Figure 26. Le modèle d'un support d'activité

6.8.4 *L'architecture cadre de DARE*

Le modèle de la Figure 26 définit l'architecture cadre de DARE. Le méta-modèle de la Figure 25 en définit la manipulation.

En effet, les classes générales de la Figure 26 correspondent aux classes de base apparaissant dans l'architecture cadre. SupportActivite, OutilGen, RoleGen et MicroRoleGen en sont les principales classes abstraites. Chacune de ces classes doit être spécialisée pour composer un support d'activité particulier.

Le méta-modèle que nous avons présenté dans la Figure 25 indique les points chauds de l'architecture cadre : il est par exemple possible d'ajouter ou de retirer des micro-rôles à un rôle. Si les classes désirées existent[1] déjà, il suffit de les lier en utilisant les méthodes de l'interface méta des classes 'composants' visées. Par exemple, si la classe TableauBlanc (une spécialisation de OutilGen) existe, elle peut simplement être ajoutée à une tâche en utilisant la méthode ajouterOutil(...) de Tache. L'architecture cadre est alors utilisé en tant que boîte noire.

A l'inverse, lorsque la classe désirée n'existe pas, il faut la définir soit en héritant directement de la classe de base, soit en héritant d'une classe existante. Là encore, l'interface méta joue un rôle prépondérant en définissant les méthodes permettant cet héritage, ainsi que celles utilisées pour la spécialisation. On pourra par exemple hériter d'un micro-rôle et le spécialiser en le liant à un nouvel outil ou en définissant de nouvelles actions. Dans ce cas, l'architecture cadre est utilisé en tant que boîte blanche.

Différents niveaux d'abstraction sont accessibles : un sujet peut par exemple modifier une tâche en manipulant des rôles[2] sans se soucier du fait qu'ils sont constitués de micro-rôles. Différents niveaux d'abstraction au niveau organisationnel sont eux aussi accessibles : une tâche est constituée d'autres tâches, qui sont des composants à part entière compréhensibles de par leur objet.

[1] Les classes peuvent préexister dans un dépôt de classes qui correspond aujourd'hui à l'image de Smalltalk.
[2] Il faut toutefois que ces rôles soient compatibles avec les autres éléments de la tâche.

6.8.5 La projection vers Smalltalk

6.8.5.1 Le choix de Smalltalk

Les mécanismes mis en place dans le noyau de DARE reposent principalement sur le MOP et requièrent à la fois de pouvoir faire de l'introspection et de l'intercession sur le système au cours de sa propre exécution. L'introspection est par exemple utilisée lorsqu'on veut lister les opérations disponibles d'un outil. L'intercession quant à elle permet d'en définir de nouvelles. Pour implémenter le noyau de DARE, il nous fallait donc un langage implémentant le MOP et supportant ces mécanismes. Nous avons choisi *Smalltalk.*

Les raisons qui nous ont poussés à utiliser Smalltalk sont diverses. Entre autres, et comme nous venons de le préciser, Smalltalk implémente le MOP et les mécanismes réflexifs qu'il offre correspondent bien à nos besoins. DARE devant reposer sur une architecture distribuée (comme nous le verrons au chapitre 7), nous avons choisi d'utiliser CORBA [48] et il nous fallait un langage et un produit supportant ce standard. C'est le cas de *Distributed Smalltalk.* Enfin, nous pouvons invoquer notre propre héritage culturel [52]...

Une question récurrente concernant ce choix est « pourquoi ne pas avoir utilisé Java ? ». Cette question est d'autant plus justifiée que Java implémente lui aussi le MOP, est compatible CORBA, et constitue le langage que nous avons nous-mêmes utilisé pour implémenter la partie 'cliente' de DARE (cf. Chapitre 7). La réponse est simple : Java n'offre pas des mécanismes réflexifs assez puissants pour supporter nos besoins, en particulier en terme d'intercession.

Dans l'approche que nous venons de présenter, il arrive souvent que la classe d'un objet en cours d'exécution soit non seulement consultée, mais aussi modifiée. Par exemple, le système est prévu pour qu'un sujet puisse ajouter ou supprimer dynamiquement des opérations dans la classe d'un outil, étendant ou limitant de ce fait les moyens de son utilisation. Ce genre de mécanismes ne peut être implémenté en Java. Java permet de charger dynamiquement des classes précompilées dans un environnement d'exécution, et ainsi, de créer des objets d'un nouveau type, mais pas de modifier la classe d'un objet en cours d'exécution.

6.8.5.2 Les principes de base

Il existe en Smalltalk plusieurs moyens de créer une interface méta modifiée. Le moyen le plus simple est certainement celui qui revient à modifier la méta-classe de base du langage, c'est-à-dire la classe Class elle-même. Ceci qui a pour effet d'étendre directement la structure et le comportement de toutes les classes qui en sont instances. Le problème dans ce cas est que la modification affecte effectivement toutes les classes du système.

Dans notre approche, nous devons définir une interface méta particulière aux différentes classes 'de base' de DARE. Smalltalk fournit un moyen simple de définir à la fois ces classes basiques et leur méta-classe respective. Il s'agit de définir des attributs et méthodes (les opérations de Smalltalk) *d'instance* de la classe, ainsi que les attributs et méthodes *de classe* de la classe. Ce qui est déclaré au niveau instance correspond à une définition classique de classe. Ce qui est déclaré au niveau classe correspond à nos méta-types dont les signatures de méthodes décrivent nos interfaces méta. Prenons un exemple simple qui nous familiarisera avec ces notions et la syntaxe de Smalltalk.

Nous créons une classe Perroquet. Une méthode d'instance de cette classe est la méthode bonjour qui a pour effet d'afficher 'bonjour' à l'écran. Cette méthode est déclarée comme suit:

```
!Perroquet methodsFor: 'parler'!          « déclaration d'une méthode d'instance »
bonjourTranscript show: 'bonjour'! !
```

Nous pouvons remarquer que la méthode bonjour est déclarée sous la classification parler de la classe Perroquet. En Smalltalk, toute classe peut définir ses propres classifications pour y ranger ses méthodes. Il est ainsi possible, par introspection de demander à une classe de renvoyer la liste de ses méthodes apparaissant dans une classification particulière. Ceci nous est très utile pour classifier les méthodes (Operation et Action) créées dans les classes de DARE par les sujets. Ainsi par exemple, lorsqu'un sujet 'inspecte' un outil, il ne voit que les méthodes intéressantes à son niveau d'abstraction (les opérations).

bonjour est une méthode d'instance de la classe Perroquet. De ce fait elle peut être déclenchée sur tout objet instance de cette classe. Il faut d'ailleurs instancier Perroquet pour qu'il affiche 'bonjour' à l'écran :

```
|unPerroquet|                    « déclaration d'un objet »
unPerroquet := Perroquet new.          « instanciation »
unPerroquet bonjour              « dit bonjour »
```

Nous voulons maintenant créer une méthode permettant à tous les perroquets (les instances de Perroquet) d'apprendre de nouveaux mots. Chaque nouveau mot correspond à ajouter une nouvelle méthode dans la classe Perroquet. Dans une programmation classique, il faudrait effectivement déclarer « à la main » ces nouvelles méthodes d'instances. De plus, avec un langage comme Java, il faudrait arrêter toutes les applications ayant des instances de Perroquet, recompiler la classe, et redémarrer les applications.

En Smalltalk, puisqu'il s'agit ici de modifier la classe Perroquet, nous déclarons une méthode apprend dans ses méthodes de classe. Cette méthode compile et ajoute dynamiquement une méthode d'instance à la classe Perroquet :

```
!Perroquet class methodsFor: 'construction'!      « déclaration d'une méthode de classe »
apprend: aString
    self compile: (aString, ' Transcript show: ''', aString, ''') classified: 'parler' asSymbol
```

Il est alors possible de s'adresser directement à la classe pour lui apprendre de nouveaux mots:

```
Perroquet apprend: 'hello'.      « invocation de la méthode de classe »
unPerroquet hello                « invocation de la nouvelle méthode d'instance »
```

En réalité, les méthodes de classe de Perroquet sont des méthodes d'instance d'une méta-classe Perroquet class. En Smalltalk, toute classe possède automatiquement une méta-classe associée. Cette méta-classe est créée par le système au moment de la création de la classe. Ainsi, lorsque nous avons créé la classe Perroquet, Smalltalk a automatiquement créé sa méta-classe Perroquet class. La méthode de classe apprend de Perroquet est donc en réalité une méthode d'instance de sa méta-classe Perroquet class.

Cet exemple simple expose certains des mécanismes réflexifs de Smalltalk que nous avons utilisé pour implémenter DARE. Chaque classe de base du système possède un ensemble d'attributs et méthodes d'instance qui décrivent la structure des objets participants à un support d'activité. Certaines de ces classes possèdent aussi des attributs et méthodes de classe qui permettent de la remodeler à l'exécution.

6.8.5.3 La projection

On trouve principalement dans le système les classes Task, Tool, Role et MicroRole qui possèdent à la fois des attributs et méthodes d'instance et de classe. Il existe donc dans le système les méta-classes Task class, Tool class, Role class et MicroRole class.

Les attributs et méthodes d'instance implémentent l'interface de base des classes de notre architecture cadre. Le niveau instance de Task, Tool, Role et MicroRole implémente respectivement les classes SupportActivite, OutilGen, RoleGen et MicroRoleGen du modèle d'un support d'activité présenté dans la Figure 26. On trouve aussi à ce niveau les classes User, Member, Operation et Action qui ne possèdent que des attributs et méthodes d'instance[1].

Les attributs et méthodes de classe implémentent l'interface méta permettant de (re)composer les classes de l'architecture cadre (les composants) et/ou de les étendre. Ainsi, les méta-classes Task class, Tool class, Role class et MicroRole class implémentent respectivement les méta-types Tache, Outil, Role et MicroRole du méta-modèle de DARE représenté Figure 25.

La Figure 27 synthétise les relations existantes entre les éléments du noyau Smalltalk et un support d'activité particulier. Cette figure ne représente que la réalisation du modèle et du méta-modèle de la tâche. Un support d'activité anActivity est instance d'une tâche spécifique ATask. Cette tâche est une sous-classe de la classe Task qui fait partie du noyau de DARE que nous avons réalisé. ATask est instance d'une méta-classe ATask class. ATask class est automatiquement créée par Smalltalk lors de la création de ATask. Task hérite de la classe Object de Smalltalk et est instance d'une méta-classe Task class qui fait elle aussi partie du noyau de DARE. Task class réalise une partie du méta-modèle de DARE et hérite de la méta-classe Class de Smalltalk.

Pour résumer, Task est la classe abstraite de notre architecture cadre. Toute nouvelle tâche hérite de Task ou d'une de ses sous-classes. Task class contient ce que nous avons précédemment appelé les *attributs de classe* et *méthodes de classe* de la classe Task. Task class est la réalisation du méta-type tâche.

Les sujets qui utilisent le support d'activité correspondant à anActivity réalisent la tâche décrite dans ATask. Pour modifier ATask, ce sont les méthodes de ATask class héritées de Task class qui sont utilisées.

[1] En réalité, ces classes possèdent aussi des attributs et méthodes de classe communs à toutes les classes de Smalltalk, du fait des propriétés du langage.

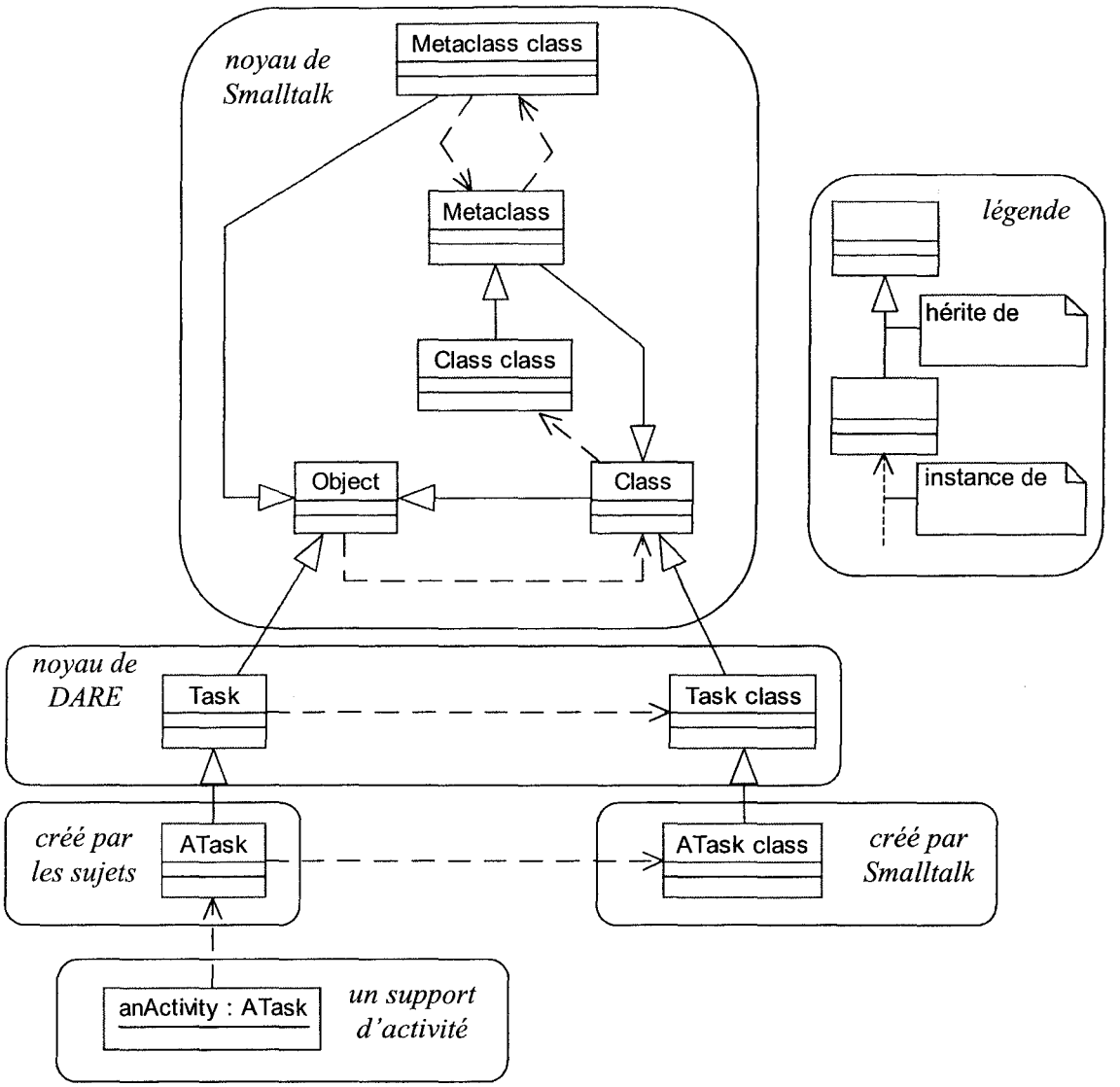

Figure 27. Du noyau Smalltalk au support d'activité (point de vue de la tâche)

La Figure 28 représente l'ensemble des classes et méta-classes principales ayant été créées pour réaliser le noyau de DARE. Les classes Member et User n'ont rien de particulier. Les classes Tool, Role, MicroRole et leur méta-classe respective Tool class, Role class et MicroRole class ont été réalisées suivant le même schéma que Task et Task class (cf. Figure 27).

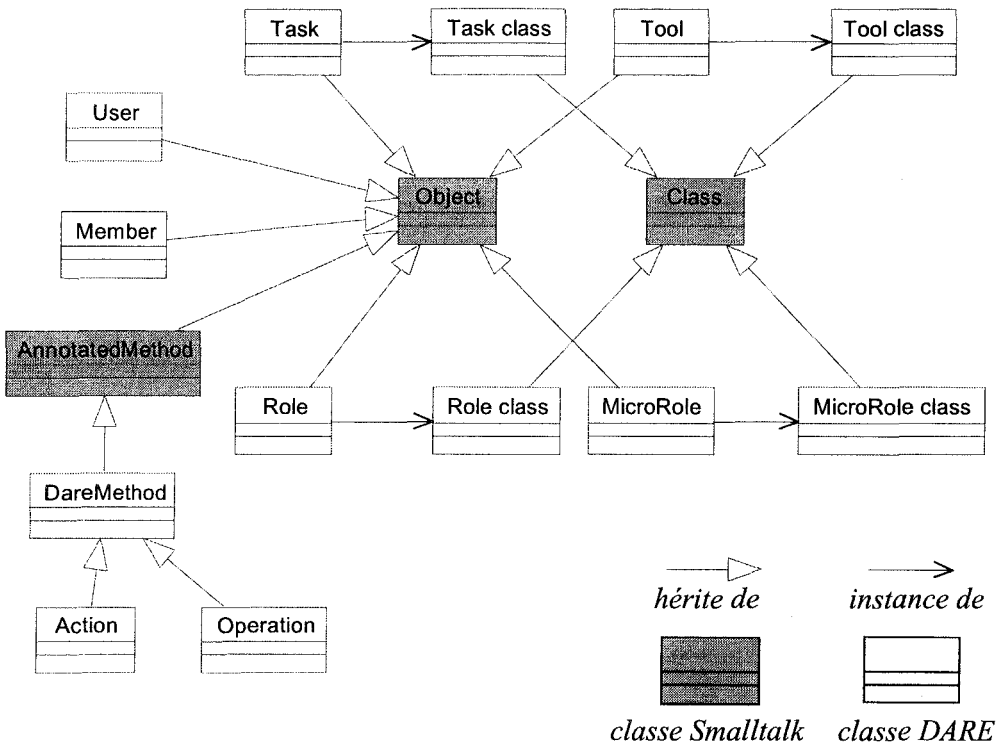

Figure 28. La réalisation de DARE dans Smalltalk

Les classes Operation et Action héritent de la classe Smalltalk AnnotatedMethod qui représente un type de méthode compilée. De ce fait, les instances d'Operation et d'Action sont elles aussi des méthodes compilées : les mécanismes d'envoi de message ou encore de lookup de Smalltalk fonctionnent donc pour les opérations sur les outils et pour les actions sur les micro-rôles. Toute action ou opération compilée contient son propre script sous forme textuelle, un nom plus compréhensible que le nom réel de la méthode, ainsi qu'une description (cf. Figure 25). Le but de cette spécialisation est de permettre aux sujets de consulter et créer des opérations et actions d'une manière plus naturelle, en ayant moins l'impression de 'programmer' le système. Pour ce faire, nous avons (entre autre) ajouté une méthode d'instance nommée methodClass dans les méta-classes Tool class et MicroRole class (donc une méthode de classe des classes Tool et MicroRole). Cette méthode revoit le type Operation pour Tool class et Action pour MicroRole class. Elle oblige ainsi le compilateur Smalltalk à réaliser les méthodes d'instance de Tool et MicroRole (et de leurs sous-classes) respectivement comme des opérations et des actions. De cette manière, un sujet ajoute bien des opérations et actions à ses outils et micro-rôles, et DARE réutilise les mécanismes fondamentaux de Smalltalk pour les

compiler et effectuer les envois de message correspondants au déclenchement d'opérations sur un outil et d'actions sur un micro-rôle.

```
Object subclass: #Task "Version (très) simplifiée de la classe Task"
          instanceVariableNames: 'members ' "attributs d'instance"

"quelques méthodes d'instance"
!Task methodsFor: 'public'!

addMember: aMember                  "ajout d'un membre dans un support d'activité"
          aMember isHere: false.
          aMember activity: self.
          self members add: aMember.
          aMember addDependent: self.              "pour les événements"
          self update: #members with: nil from: self.
          ^self!

createMember: aUser      withRole: aRole  "création d'un membre"
          | member |
          member := Member new.
          member userDef: aUser.
          member setRole: aRole.
          member connector: (ORBObject factoryFinder createObject: DAREMember).
          member connector member: member.                "pour CORBA"
          self addMember: member.
          ^member!

removeMember: aMember "retrait d'un membre"
          self members remove: aMember.
          aMember removeDependent: self.
          self update: #members with: nil from: self.! !
"..."

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

Task class
          instanceVariableNames: 'tools roles subTasks '! "attributs de classe"

"quelques méthodes de classe"
!Task class methodsFor: 'public'!

object: aString                              "l'objet de l'activité défini dans une tâche"
          self comment: aString.
          self update: #object with: nil from: self.!
"..."
addRole: aRole              "aRole est une classe de roles"
          self roles at: aRole name put: aRole.
          aRole addDependent: self.
          self update: #roles with: nil from: self.!
"..."
```

Figure 29. Les niveaux instance et classe de la classe Task de DARE (simplifiée) dont héritent toutes les tâches définissant un support d'activité

La Figure 29 représente la définition, en version très simplifiée, de la classe Smalltalk Task (et Task class qui y est associée). Cette classe est la base de toute tâche pouvant être construite dans DARE. De ce fait, toute tâche définie par des sujets hérite de cette classe. On distingue

bien deux parties dans cette définition : l'interface de base définie au niveau instance et son interface méta définie au niveau classe. On voit en particulier que tout support d'activité (instance de Tache) peut accueillir un ensemble de membres. Ces membres utiliseront les outils et joueront les rôles définis au niveau classe.

Nous avons aussi fait apparaître la possibilité de définir ou redéfinir l'objet d'une activité. La définition de l'objet est accessible par l'interface méta et est donc défini au niveau des méthodes de classe. Pour stocker cet objet, nous utilisons un attribut comment commun à toute classe Smalltalk (défini dans la méta-classe Class). Cet exemple montre d'une manière simple comment, pour appréhender l'ensemble du système, il serait nécessaire de comprendre un grand nombre de mécanismes définis dans les classes fondamentales du langage Smalltalk. Ceci explique que nous ne présentions qu'une version simplifiée des fondements de l'implémentation de DARE. Toutefois, le travail d'un utilisateur (informaticien ou sujet) est amplement simplifié en lui permettant d'utiliser l'interface méta 'composite[1]' de l'architecture cadre.

La Figure 30 représente la classe Tool, une autre des classes de base de DARE possédant une interface méta. L'étude de cette classe est intéressante car elle expose plus clairement la mise en œuvre de l'introspection et de l'intercession dont nous avons précédemment parlé. En effet, les sujets sont amenés à intégrer des outils dans leurs support d'activités et donc dans leurs tâches. Pour utiliser un outil, ou pour définir des micro-rôles sur celui-ci, il est nécessaire que les sujets puissent examiner ses opérations. Ceci est réalisé par une méthode de classe nommée operations. Dans notre implémentation, chaque opération correspond à une méthode d'instance de la classe. Examiner une opération correspond donc à faire de l'introspection sur la classe en listant toutes les méthodes rangées sous la classification subjects.

Enfin, les sujets étant amenés à modifier leurs outils et donc spécifier les opérations qui leur sont applicables, une des méthodes de classe de Tool est addOperation qui permet d'ajouter une méthode d'instance à la classe visée (une sous-classe de Tool) à partir d'un nom et d'un script spécifiés[2] par un sujet. addOperation modifie donc la classe sur laquelle elle a été invoquée : c'est de l'intercession. Toute opération ainsi créée est alors directement accessible aux sujets dans leur support d'activité.

[1] Il y a une interface méta par composant de base. Les nouveaux composants héritent de ces composants basiques et donc de l'interface méta générique.
[2] La construction d'un script par un sujet sera détaillée dans le chapitre suivant.

```
DAREClassWithTarget subclass: #Tool
"..."

!Tool methodsFor: 'subjects'!

        "ici seront classées les methodes (operations) définies par les sujets"
        "en utilisant l'interface méta, c'est-à-dire, les méthodes de classe"

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

Tool class
        instanceVariableNames: 'codeBase archives params '!          "les attributs de classe"

"des méthodes de classe :"
!Tool class methodsFor: 'public'!

operations          "pour récupérer les opérations, y compris celles héritées : introspection"
        | methods |
        self = Tool
                ifTrue: [methods := Dictionary new]
                ifFalse: [methods := self superclass operations].
        (self organization listAtCategoryNamed: 'subjects' asSymbol)
                do: [:selector | methods at: selector put: (self compiledMethodAt: selector)].
        ^methods! !

addOperation: aName From: aScript          "pour ajouter des opérations : intercession"
        |method|
        method := self compile: (aName, aScript) classified: 'subjects' asSymbol.
        (self compiledMethodAt: method) initialize.   "pour les méta informations"
        self changed: #operations.
        ^method!

"..."
```

Figure 30. La classe Tool de DARE (simplifiée) dont héritent tous les outils: utilisation de l'introspection et de l'intercession

6.9 En résumé

Les points principaux à retenir de ce chapitre sont listés dans le Tableau 7 ci-dessous. Du point de vue de l'informaticien, DARE fournit à ses utilisateurs non seulement un support à leurs activités, mais encore son propre environnement de programmation. Il est toutefois très important de noter que cet environnement de programmation met en œuvre un langage fondé sur les concepts et mécanismes identifiés par l'AT. Ainsi, du point de vue des sujets qui accèdent à la malléabilité de DARE, il ne s'agit pas de (re)programmer leur application, mais de (re)spécifier la tâche associée à leur support d'activité.

- L'implémentation ouverte répond bien à nos problèmes en terme de malléabilité.
- DARE adopte une approche mêlant une architecture cadre et l'implémentation ouverte.
- Le modèle de l'architecture cadre fournit un modèle générique de support à l'activité fondé sur les concepts introduits par la Théorie de l'Activité et leurs interrelations.
- L'implémentation ouverte permet d'offrir aux sujets une interface de manipulation des modèles de support d'activité à un niveau d'abstraction fondé sur les concepts et mécanismes identifiés par la Théorie de l'Activité.
- La malléabilité de DARE est hiérarchique et supporte l'intégration et l'extension.
- Le Meta-Object Protocol a permis de faciliter l'implémentation de DARE qui bénéficie des mécanismes d'un langage réflexif.

Tableau 7. A propos de l'implémentation du noyau de DARE

Il nous reste maintenant à présenter les moyens que nous avons mis en œuvre pour permettre aux sujets d'accéder à cet environnement réflexif de support aux activités collectives. Ces moyens reposent principalement sur une architecture distribuée à la topologie en constante extension. Cette architecture est présentée dans le chapitre 7.

Chapitre 7

Architecture distribuée

Nous avons présenté dans le chapitre 6 les concepts, les techniques logicielles et la réalisation du noyau de DARE. Ce chapitre présente les moyens qui permettent aux utilisateurs d'accéder à l'environnement DARE. Dans un premier temps, nous proposons une solution technique pour supporter notre architecture distribuée. Nous présentons ensuite les moyens mis en œuvre pour que DARE remplisse le cahier des charges proposé au chapitre 1, et enrichi au fil des autres chapitres. En résumé, DARE doit être un environnement de Travail Coopératif Assisté par Ordinateur (TCAO) global, intégré, malléable, supportant la mobilité des utilisateurs, et au plus proche des fondements de l'activité humaine proposés par la Théorie de l'Activité (AT).

7.1 Un accès standard par Internet

La fonction principale d'un collecticiel est de fournir un support informatique à l'activité coopérative. Dans le cas où les sujets sont physiquement distants, une des fonctions du collecticiel est de permettre aux sujets de participer à distance à leur activité coopérative. Grâce aux technologies sans cesse plus puissantes des réseaux informatiques, le collecticiel permet de créer un environnement virtuellement partagé mais physiquement distribué. Cette distribution physique implique que les sujets aient un accès au réseau leur permettant de se 'connecter' à l'environnement virtuel de TCAO.

Nous pensons qu'un autre objectif, est de favoriser la mobilité des sujets. Autrement dit, nous voulons non seulement permettre aux sujets de travailler à distance, mais aussi leur permettre de se connecter à partir de n'importe quel site client banalisé. La mise en pratique de cette démarche implique que n'importe quel ordinateur possédant une connexion réseau et une configuration standard permette au sujet d'accéder à son environnement de TCAO. La conséquence majeure d'une telle condition est de proscrire toute installation préalable de logiciels spécifiques dédiés au collecticiel sur le poste client. De telles remarques avaient déjà été formulées dans le chapitre 1.

Aujourd'hui, nous pouvons considérer que tout poste client connecté au réseau possède au minimum un navigateur Internet. La plupart des navigateurs contiennent une machine virtuelle Java leur permettant d'exécuter des applets. Rappelons que les applets sont des applications Java qui peuvent être téléchargées sur le poste client lorsque l'utilisateur en a besoin. En d'autres termes, sous forme d'applets, les outils nécessaires pour agir dans l'environnement de TCAO peuvent être rapatriés sur un poste client banalisé. Ils ne demandent aucune installation préalable, si ce n'est celle d'un navigateur compatible Java.

Les applets fournissent aux sujets un moyen d'agir dans l'activité coopérative distribuée sur le réseau. Il est important que ces applets soient en permanence connectées au reste de l'activité. En effet, pour créer l'environnement virtuel de TCAO, les événements pouvant intervenir dans le support d'activité comme l'arrivée d'un nouveau participant, doivent être répercutés sur les postes des différents sujets connectés. Il s'agit là de la prise en compte de la notion de co-présence bien connue du TCAO [28] [6].

L'architecture de DARE a été conçue pour répondre à ces différents enjeux. Les sujets peuvent accéder au support d'activité qui les intéresse en utilisant un poste banalisé connecté au réseau. D'une manière générale, un navigateur Internet permet au sujet de se connecter à un *serveur d'activités* qui lui fournit les *applets outils* dont il a besoin pour réaliser sa tâche. Ces applets offrent au sujet un moyen d'agir dans le support d'activité (cf. Figure 31). Le serveur d'activités contient les différents éléments présentés dans le chapitre 6. Comme nous l'avons vu, il est réalisé en Smalltalk et contient le *noyau*[1] de DARE. Tous les composants (tâches, outils, rôles et micro-rôles) spécifiés par les sujets y sont stockés et mis en œuvre.

[1] Le noyau de DARE contient l'implémentation du modèle générique de support d'activité et son méta modèle.

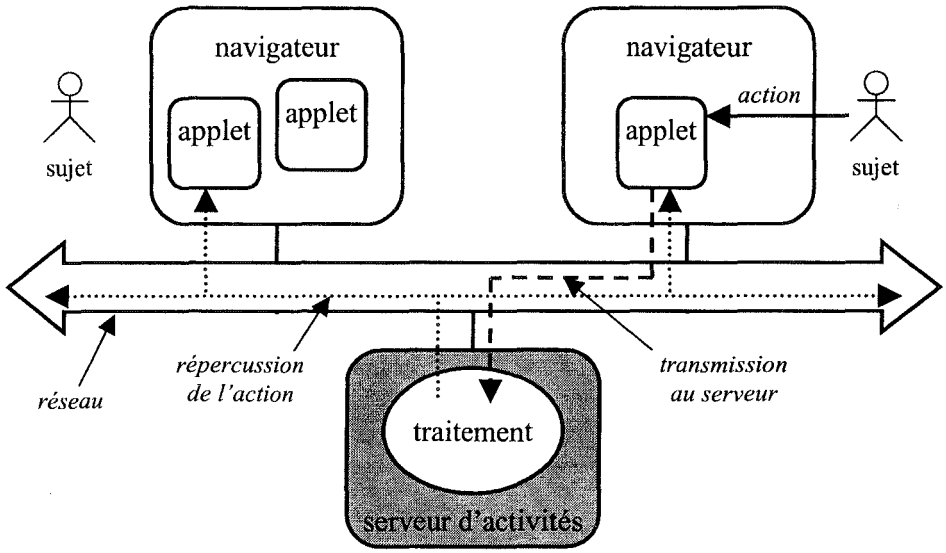

Figure 31. La répercussion des actions des sujets au travers du réseau

DARE repose donc sur une architecture distribuée : les acteurs principaux en sont un serveur d'activités et un nombre à priori quelconque de postes clients. Pour implémenter cette architecture distribuée, nous nous sommes appuyés sur l'architecture logicielle définie par le consortium international OMG (*Object Management Group*) : l'architecture CORBA (*Common Object Request Broker Architecture*).

7.2 Une solution technologique : CORBA

7.2.1 Généralités

Dans le but de faciliter le développement d'applications réparties, l'OMG s'est fixé pour objectif de définir un standard pour l'intégration d'applications distribuées. Un des objectifs secondaires est de mettre en œuvre les propriétés qui ont fait le succès de l'approche objet comme l'encapsulation, l'héritage et le polymorphisme. Notons au passage que ces propriétés sont aussi celles qui sont mises en avant dans la création et le déploiement de composants[1] logiciels tels que nous les avons déjà présentés.

Le premier travail réalisé par l'OMG fut la définition de l'OMA (*Object Management Architecture*) [101]. Elle répond au problème introduit par la diversité des modèles objets

[1] L'OMG travaille d'ailleurs actuellement sur la définition d'un modèle de composant pour CORBA.

existants, dans les langages de programmation ou dans les bases de données orientées objet par exemple. L'OMA spécifie un modèle objet et une architecture d'intégration qui sont utilisés dans toutes les technologies de l'OMG.

7.2.2 CORBA, l'IDL et les règles de projection

La spécification de CORBA suivit directement la définition de l'OMA. CORBA spécifie le bus à objets répartis qui permet l'interopérabilité entre des objets pouvant être réalisés dans différents langages d'implémentation, et implantés sur des systèmes d'exploitation hétérogènes. A partir de cette spécification, il est possible d'implémenter un bus CORBA ou ORB (*Object Request Broker*) qui répond au standard de l'OMG. Il existe aujourd'hui de nombreux ORBs dont les plus connus sont Orbix, Visibroker, ou encore Distributed Smalltalk.

Dans CORBA, l'interopérabilité entre objets hétérogènes est rendue transparente par l'introduction d'interfaces spécifiées en OMG-IDL (*Interface Definition Language*). Ces interfaces décrivent le service fourni par un objet CORBA indépendamment de son implémentation. Le passage d'une interface IDL vers une interface dans un langage d'implémentation spécifique est réalisable grâce à *des règles de projection* normalisées et définies par l'OMG.

Nous avons déjà mentionné que la partie cliente des supports d'activité de DARE est réalisée en Java, alors que la partie serveur l'est en Smalltalk. L'architecture distribuée de DARE met donc en œuvre à la fois des objets implémenté en Java (des applets), et des objets implémentés en Smalltalk. Les règles de projection de l'IDL sont à ce jour bien définies pour ces deux langages.

La Figure 32 représente le mécanisme général des interactions entre objets Java et objets Smalltalk qui apparaissent dans DARE. Chaque objet possède une interface IDL qui le décrit et qu'il implémente. L'implantation d'un tel objet sur un ORB en font un objet CORBA qui offre des services à tout autre objet du bus. Dans DARE, les objets Java sont les applets instanciées dans les navigateurs des postes clients. Les objets Smalltalk sont ceux qui font partie du serveur d'activités. Ici, les objets sont généralement à la fois clients et serveurs. En particulier, les applets Java font appel aux services proposés par les objets du serveur d'activités qui gèrent une représentation globale des supports d'activité. A l'opposé, les objets

Smalltalk du serveur font appel aux applets Java pour qu'elles se mettent à jour et offrent une représentation cohérente du support d'activité auquel elles sont liées.

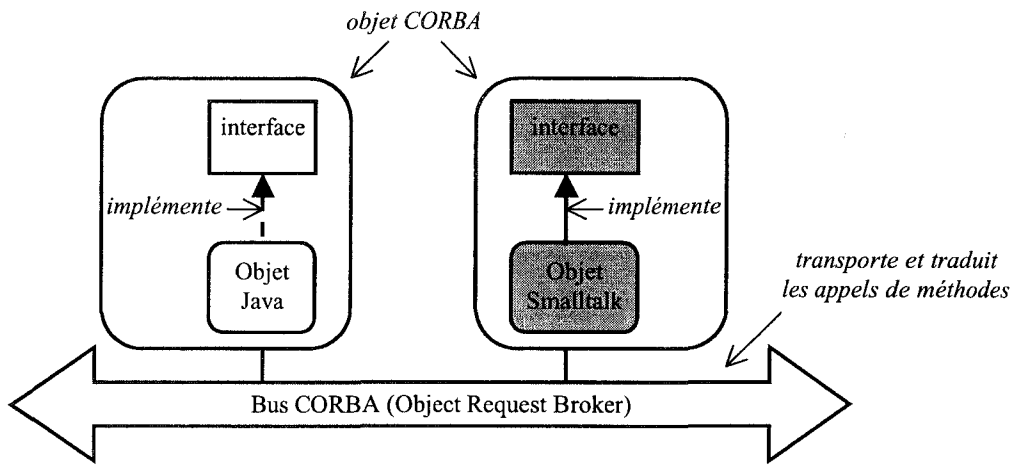

Figure 32. L'interaction entre objets Java et Smalltalk par CORBA

7.2.3 L'interopérabilité

Un ORB permet l'interaction entre des objets hétérogènes. Toutefois, jusqu'à CORBA 2.0, un important problème d'hétérogénéité entre les ORBs eux-mêmes subsistait. En effet, les objets hétérogènes étaient capables d'interagir au sein d'un ORB, mais incapables d'interagir avec les objets d'un ORB différent. Par exemple, les objets CORBA d'Orbix ne pouvaient faire appel aux objets CORBA de Visibroker. CORBA 2.0 résout ce problème en introduisant le support d'interopérabilité entre ORBs nommé GIOP (*General Inter-ORB Protocol*). En particulier, le mécanisme IIOP (*Internet Inter-ORB Protocol*) de GIOP définit la communication basée sur TCP/IP entre différents ORBs.

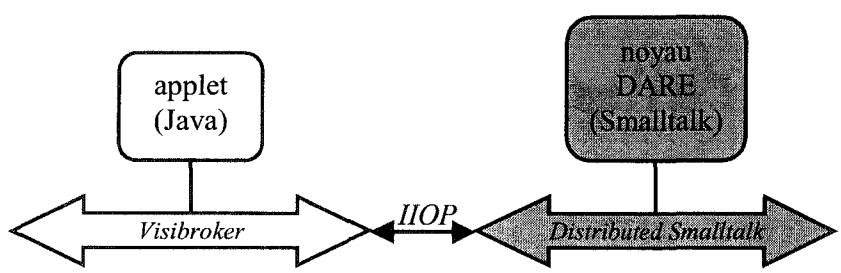

Figure 33. L'interaction entre les deux ORBs utilisés dans DARE

Dans DARE, l'IIOP possède un rôle important car certains des objets Java qui composent les applets sont implantés dans l'ORB Visibroker utilisable sous Netscape, alors que les objets Smalltalk sont implantés dans l'ORB Distributed Smalltalk (cf. Figure 33).

7.2.4 Conclusion sur les choix technologiques

Nous avons brièvement présenté quelques-uns des principes sous-jacents aux architectures distribuées utilisant CORBA, en insistant sur quelques points directement liés à la réalisation de DARE. Comme nous l'avons présenté, DARE utilise CORBA pour plusieurs raisons : un niveau d'abstraction élevé permettant de réaliser plus facilement des architectures distribuées ; la possibilité d'interactions entre des objets implémentés dans des langages hétérogènes, en particulier Java et Smalltalk.

CORBA présente de nombreuses autres propriétés particulièrement intéressantes, comme les services qui y sont attachés, la liaison dynamique, etc. Nous avons cependant choisi de limiter notre présentation aux principes généraux essentiels à la compréhension de notre réalisation. Nous n'avons pas non plus présenté les problèmes rencontrés dans la mise en œuvre d'une telle architecture, comme ceux liés à la sécurité mise en place dans Java au niveau des applets. Cette sécurité rend difficile la communication entre objets distants et ne peut être outrepassée que par la mise en place de mécanismes comme le *HTTP Tunnelling* ([133], p 50) (que nous avons utilisé dans notre prototype), ou encore la signature des applets impliquées. De larges descriptions et études de CORBA peuvent être trouvées dans [48].

7.3 Le support d'activité

7.3.1 La connexion au support d'activité

De manière à pouvoir interagir dans un support d'activité, il faut que les sujets récupèrent leur environnement de travail. La Figure 34 présente les moyens mis en œuvre dans DARE pour que les sujets accèdent au support d'activité. Avec son navigateur Internet, le sujet se connecte tout d'abord à un serveur HTTP (Hyper Text Transfer Protocol). Ce serveur HTTP renvoie au navigateur un document HTML (Hyper Text Mark-up Language) contenant les informations nécessaires à l'instanciation de l'environnement DARE. Ce document a été préalablement généré par le noyau de DARE (cf. Figure 35). Il contient l'URL et les

paramètres d'une applet nommée *applet activité.* Cette applet doit fournir au sujet une représentation personnalisée du support d'activité auquel elle est associée.

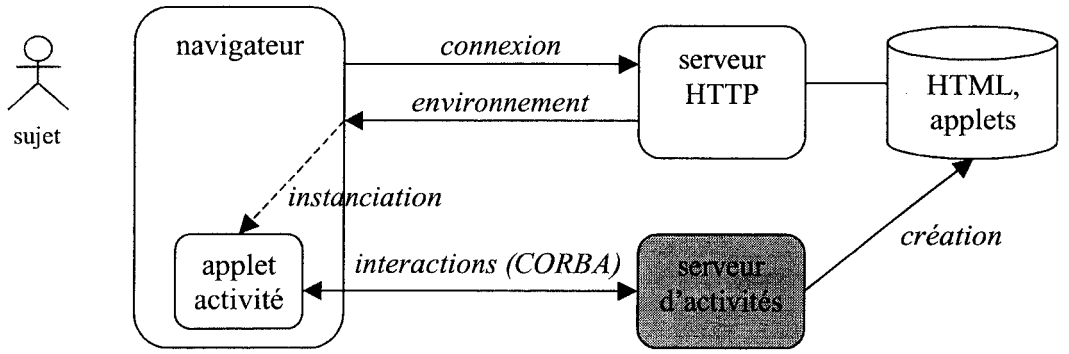

Figure 34. L'accès à DARE par un navigateur Internet compatible Java

```
<html>
<body>
<APPLET
 CODEBASE = "http://trg49:9090/Activity"
 CODE     = "ActivityApplet.class"
>
<param name=IOR value="IOR:000000000000001649444c3.....1000000">
<param name=USE_ORB_LOCATOR value = true>
<param name=ORBgatekeeperIOR value=gatekeeper.ior>
</APPLET>
</body>
</html>
```

Figure 35. Un exemple de fichier HTML pour une connexion à un support d'activité. *Le paramètre IOR est une référence d'objet CORBA qui représente un support d'activité dans le noyau de DARE. Les autres paramètres servent à configurer le navigateur lui-même (il s'agit ici de Netscape Communicator 4.07) pour qu'il utilise un bus CORBA spécifique.*

Une fois rapatriée et instanciée dans le navigateur, l'applet activité se connecte au serveur d'activités, ou plus précisément, au support d'activité qu'elle représente. La connexion est réalisée par la création dans l'applet d'une référence sur un objet CORBA qui représente le support d'activité. Le type de cet objet est CorbaActivite. Il possède une interface IDL qui décrit l'ensemble de ses services. Ceux-ci permettent à l'applet activité d'afficher un grand nombre d'informations sur son support d'activité. Ces informations concernent par exemple

les sujets impliqués dans l'activité (leur rôle, s'ils sont actuellement connectés, etc.). En particulier, un des services de CorbaActivite permet au sujet de récupérer son environnement de travail personnalisé.

7.3.2 La construction de l'environnement du sujet

Pour permettre au sujet d'accéder à sa représentation personnalisée du support d'activité, l'applet activité lui demande de s'identifier. Les informations obtenues sont transmises au support d'activité qui recherche le membre correspondant et renvoie la référence d'un objet CORBA qui le représente. Le type de cet objet est CorbaMembre.

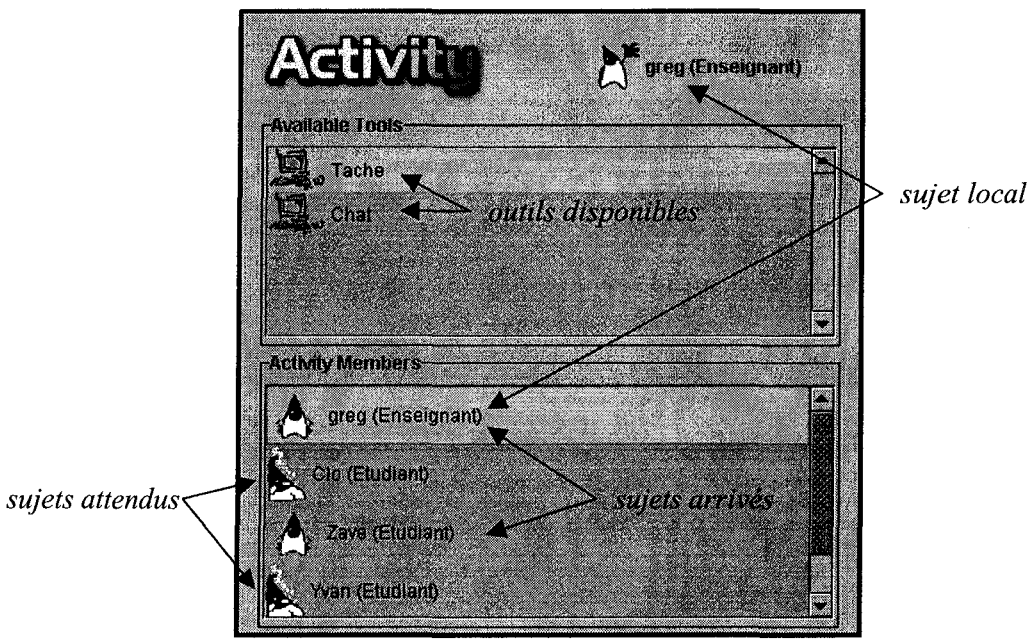

Figure 36. Un exemple d'applet activité

La Figure 36 représente un exemple d'applet activité que nous avons développée pour cette première version de DARE. Le membre (ou sujet) identifié dans cette applet s'appelle 'greg'. Il voit l'ensemble des outils qui lui sont accessibles en fonction de son rôle (Enseignant), ainsi que la liste des membres (sujets) de l'activité qui sont attendus ou déjà arrivés. Il est à noter que les notions de *sujets attendus* et *sujets arrivés* sont classiquement utilisées dans les collecticiels synchrones. Rappelons toutefois que DARE s'intéresse à la fois aux activités synchrones et asynchrones. Le type d'activité supportée dépend principalement de son objet

et des outils qu'elle met en œuvre : ces outils peuvent être des outils synchrones ou asynchrones.

Chaque outil du support d'activité correspond à un objet du type CorbaOutil et le rôle d'un membre à un objet du type CorbaRole. Le rôle permet d'accéder à l'ensemble de ses micro-rôles (du type CorbaMicroRole). Chaque micro-rôle peut renvoyer la liste des actions disponibles pour le membre qui le joue. Les services offerts par ces objets CORBA sont spécifiés dans leur interface IDL respective. La Figure 37 est un exemple d'une telle interface.

```
typedef sequence<CorbaOutil> TableauOutil ;

interface CorbaMembre{
  void setLogin(in string login) ;
  string getLogin() ;
  void setPassword(in string pwd) ;
  string getPassword() ;
  TableauOutil getOutils() ;
  void setRole(in CorbaRole role) ;
  CorbaRole getRole() ;
...
} ;
```

Figure 37. L'interface IDL d'un objet CorbaMembre

7.3.3 *Le fonctionnement de l'applet activité*

La Figure 38 représente les liens et interactions qui existent entre une applet activité et son support d'activité dans le serveur d'activités. Un support d'activité est représenté sur le serveur par un objet supportActivite instance d'une sous-classe de la classe Task. Dès que supportActivite est instancié, un objet proxy (instance de CorbaActivite), ainsi qu'un objet canal (instance de CorbaCanal) sont créés. L'objet proxy est un objet CORBA du serveur d'activités permettant aux objets distants d'interagir avec support d'activité. CorbaCanal correspond à un canal d'événements[1] auquel des CorbaEcouteur s'enregistrent pour être informés de tout événement publié sur ce canal. Comme nous l'avons vu précédemment, le noyau DARE utilise l'IOR de l'objet proxy pour générer le fichier HTML de connexion au support d'activité. L'applet activité utilise cet IOR pour créer sa reference sur l'objet proxy. Il faut souligner que plusieurs applets activité (une par sujet connecté) référencent le proxy, ce qui explique la mise en place du canal d'événements auquel les applets peuvent s'abonner : une

[1] Nous avons du réimplémenter un mécanisme d'événements en raison des problèmes d'incompatibilité entre les services d'événements des bus CORBA Distributed Smalltalk et Visibroker.

méthode du proxy invoquée par une applet activité particulière peut générer des événements qui 'préviennent' les autres applets activité connectées.

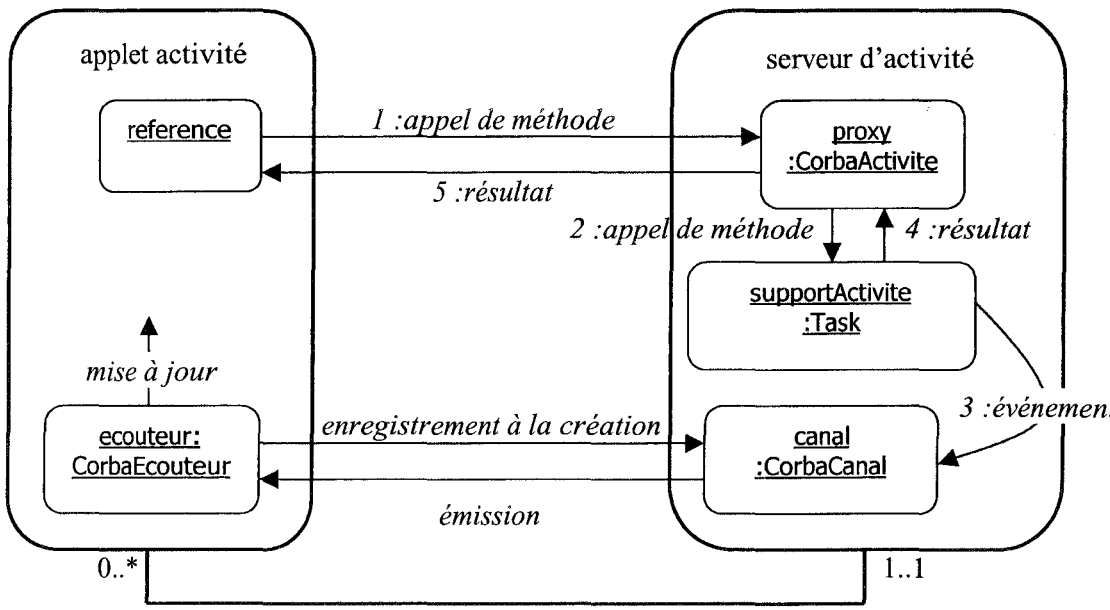

Figure 38. Interactions entre une applet activité et son support d'activité

Une méthode invoquée sur la reference dans l'applet activité est transmise par les bus CORBA au proxy (1) qui invoque les méthodes d'instance adéquates sur le supportActivite (2). Ces méthodes peuvent générer des événements sur le canal (3). Un résultat est éventuellement retourné au proxy (4) qui le retransmet à l'applet (5).

Nous pouvons noter que le premier appel de méthode réalisé par une applet activité correspond à demander au proxy une référence sur le canal. Le résultat de cette méthode permet à l'applet activité d'enregistrer un ecouteur dans le canal d'événements de son support d'activité.

L'ajout d'un membre dans un support d'activité constitue un autre exemple d'invocation de méthode. L'interface IDL de proxy définit l'opération IDL ajouterMembre(...). Une applet activité qui permet à un sujet d'ajouter un membre (cf. Figure 39) invoque cette méthode sur sa reference. La requête est transmise à supportActivite qui génère des événements alertant toutes les applets concernées qu'un nouveau membre est attendu dans le support d'activité.

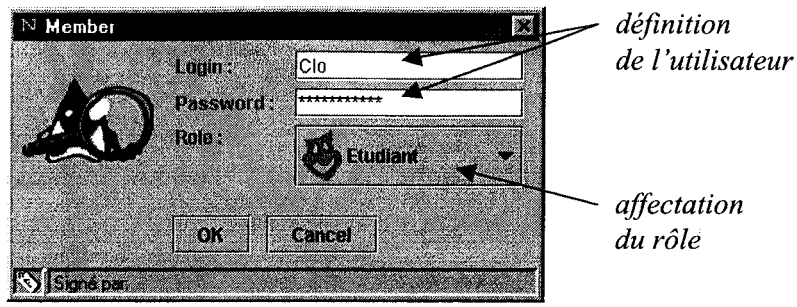

Figure 39. Exemple de (re)définition d'un membre dans une activité. *Un* Membre *est une association d'un* Utilisateur *et d'un* Role. *Le rôle (ici* Etudiant*) est une sous-classe de la classe* Role *du noyau de DARE.*

7.3.4 L'instanciation des outils

De manière à ce que le sujet puisse réaliser sa tâche, il faut instancier ses outils dans son navigateur. Ces outils peuvent tout aussi bien être une calculette qu'un client pour une audio conférence, ou encore un tableau blanc partagé. Dans notre approche (un accès par Internet supportant la mobilité des sujets), tous ces outils, aussi divers soient-ils, ont un point commun : ce sont des applets Java instanciées dans le navigateur par l'applet activité. La Figure 40 représente les liens existants entre les principaux types d'éléments d'un support d'activité distribué.

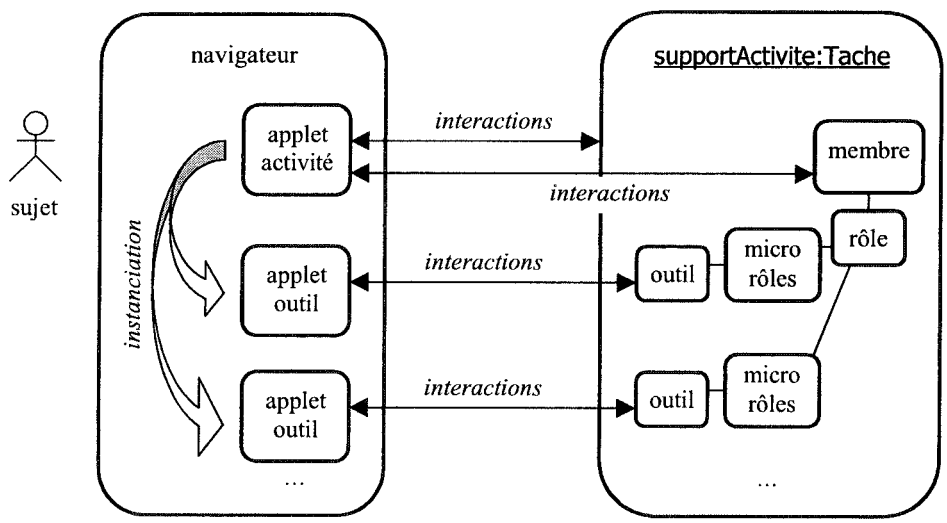

Figure 40. L'instanciation des outils d'un sujet et leurs liens avec leur support d'activité

Un autre point important est que les outils doivent être utilisés par les sujets en fonction de leur rôle. Le rôle d'un sujet, au travers de ses micro-rôles, doit déterminer les actions qu'il peut exécuter. Enfin, l'action d'un sujet peut avoir des répercussions sur ses autres outils, ainsi que sur les outils des autres sujets. Ces droits et conséquences sont gérés par le support d'activité et définis dans la tâche. De ce fait, chaque outil doit posséder une connexion au support d'activité, ou plus précisément, avec l'objet qui le représente dans la collection d'outils du membre. Comme l'indique la Figure 40, il doit donc exister des interactions entre les applets outil instanciées dans un navigateur et les objets qui les représentent dans un support d'activité sur le serveur.

Ce dernier point pose un problème non négligeable. En accord avec l'AT, nous voulons permettre aux sujets de définir et/ou de transformer leurs propres activités coopératives en définissant eux-mêmes leurs outils et rôles. Nous ne croyons pas que les sujets vont développer eux-mêmes les applets outil dont ils peuvent avoir besoin pour leurs activités. Il faut donc leur permettre d'intégrer dans l'environnement des applets existantes et n'ayant pas spécialement été développées pour DARE !

7.4 Intégration de composants applets

Comme nous venons de le préciser, il existe une contrainte imposée aux outils pouvant être intégrés dans DARE : ils doivent tous posséder une partie cliente sous forme d'une applet Java. Cette contrainte provient à la fois 1) du choix que nous avons fait en terme de mobilité des utilisateurs (utilisation d'un simple navigateur) et 2) des propriétés des applets.

7.4.1 Les composants applets

Une applet est un composant logiciel pouvant être téléchargé du réseau et instancié dans un environnement d'exécution sur un poste client (un navigateur Internet par exemple). En utilisant le mécanisme d'introspection présent dans Java (cf. chapitre 6), il est possible de récupérer les méthodes d'une applet inconnue, pour pouvoir ensuite les invoquer. Ces mécanismes sont identiques à ceux mis en place dans l'utilisation des JavaBeans (qui utilisent principalement l'introspection ainsi qu'une convention pour les noms des méthodes).

Une autre propriété intéressante des applets est de pouvoir faire appel à leur environnement d'exécution (i.e. le navigateur) pour récupérer des informations. Une référence sur ce contexte d'exécution est obtenue grâce à la méthode java.applet.AppletContext getAppletContext() de

Applet. Ceci offre le moyen pour une applet exécutée dans une page HTML de récupérer la liste des autres applets présentes (grâce à la méthode java.util.Enumeration getApplets() de AppletContext). Ainsi, une applet chargée dans une page HTML peut récupérer dynamiquement une référence sur une autre applet, lui demander la liste de ses méthodes, et même la 'piloter'. Nous allons voir que ces mécanismes nous sont d'une grande utilité pour l'intégration d'outils existants dans DARE.

7.4.2 Lier une applet à DARE

Nous définissons une *applet liée* comme une applet disponible sur le réseau et référencée[1] par un outil (une classe ou composant outil) dans le serveur d'activités. L'applet liée est la partie cliente de l'outil. Elle peut être instanciée dans le navigateur du sujet et pilotée par l'instance d'outil correspondante dans la liste des outils du membre dans un support d'activité.

Le problème est que nous voulons créer un lien fin entre l'environnement de TCAO et une applet n'ayant pas forcément été développée dans ce but. Il faut pourtant que toute opération déclenchée par un micro-rôle sur un outil du serveur d'activités soit directement répercutée sur l'applet Java liée correspondante dans le navigateur du sujet.

La définition des liens entre les méthodes de l'applet et les opérations de l'outil auquel elle est liée dans un support d'activité est assez complexe. Il faut tout d'abord récupérer les méthodes de l'applet désirée. En effet, ces méthodes ne sont pas connues puisque l'applet peut être intégrée par un sujet au cours de l'exécution du support d'activité. Une fois les méthodes récupérées, il est alors possible d'ajouter des opérations à l'outil qui référence l'applet (i.e. des méthodes d'instances d'une sous-classe de Outil au niveau conceptuel ou Tool au niveau implémentation). Chaque opération correspond à un script d'invocation de méthodes sur l'applet liée.

[1] L'attribut url de la classe d'outils contient la valeur de l'URL de l'applet liée.

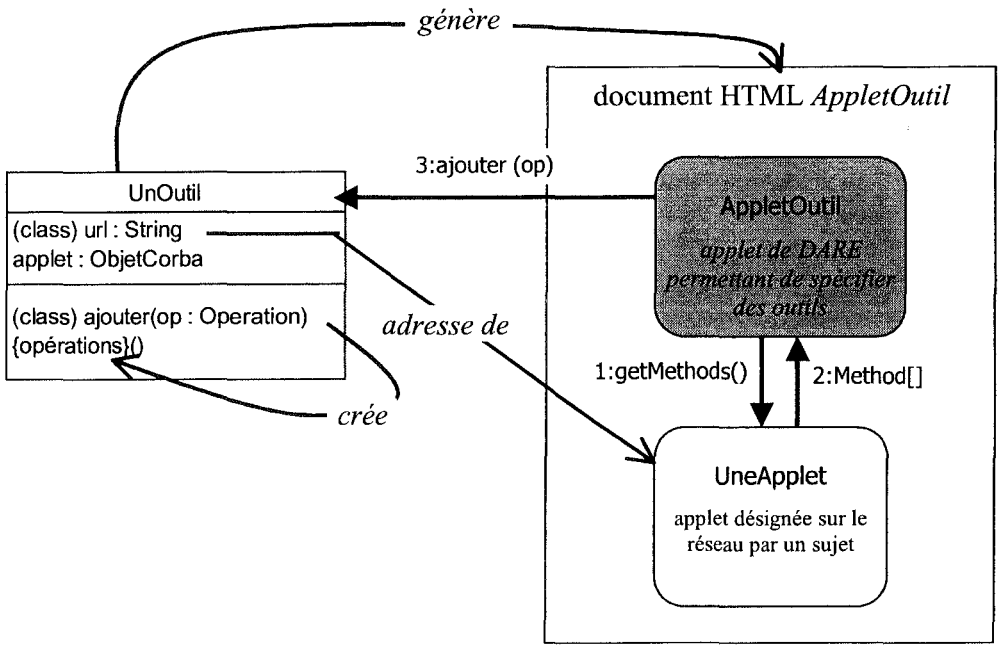

Figure 41. La création d'opérations dans une classe d'outils

La Figure 41 schématise les moyens mis en œuvre dans DARE pour créer une opération dans une classe d'outils nommée UnOutil et liée à une applet UneApplet. Pour ce faire, un document HTML est généré à partir d'un modèle générique. Le modèle contient l'URL d'une applet nommée AppletOutil, construite dans le cadre de DARE et utilisée pour créer ou modifier un outil du serveur d'activités (cf. Figure 42). Grâce à ses paramètres, AppletOutil se crée une référence d'objet CORBA qui lui permet d'appeler les méthodes de classe de UnOutil. Le document contient de plus l'URL de l'applet liée UneApplet. Lorsque ce document est chargé dans le navigateur du sujet, AppletOutil se crée une référence sur UneApplet. Cette référence permet à AppletOutil de récupérer la liste des méthodes de UneApplet (1) & (2). AppletOutil présente cette liste au sujet qui peut alors définir des opérations pour UnOutil. Lorsqu'une opération est validée par le sujet, AppletOutil déclenche la méthode de classe ajouter(...) de UnOutil, ce qui entraîne la création d'une nouvelle méthode d'instance (l'opération spécifiée) pour cette classe (3).

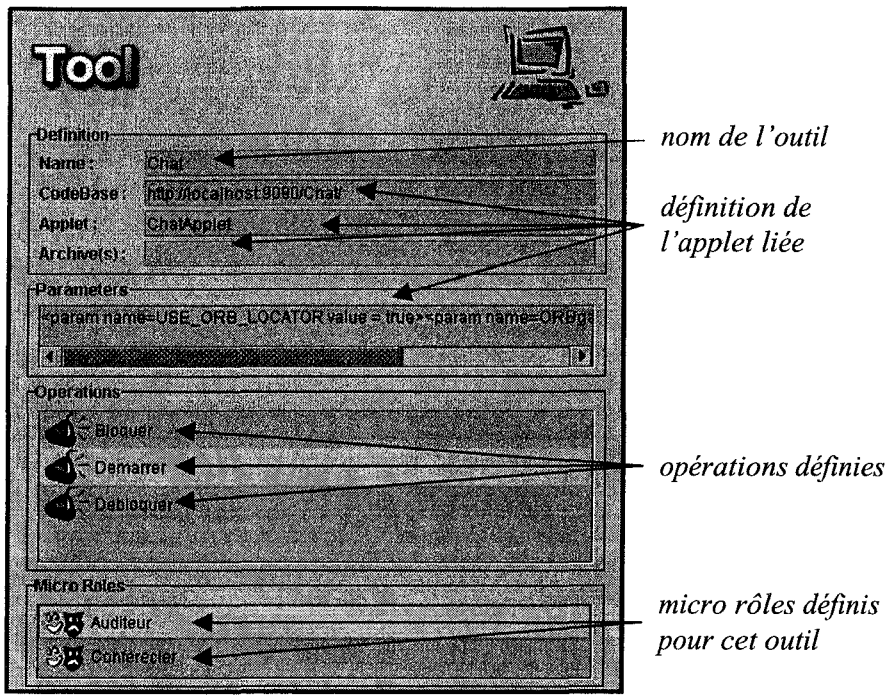

Figure 42. Un exemple d'AppletOutil : liaison de ChatApplet à une tâche

La Figure 42 est un exemple d'AppletOutil. Cette applet est utilisée par les sujets qui en ont le droit pour créer ou modifier des outils dans leur support d'activité. On trouve dans cette applet le nom de l'outil et les informations nécessaires à l'instanciation de l'applet liée (URL, paramètres, etc.). On trouve aussi la liste des opérations ayant été définies pour cet outil. Il est possible de créer de nouvelles opérations, de les modifier ou encore de les supprimer. Pour des raisons pratiques, nous avons ajouté dans AppletOutil la liste des micro-rôles associés à l'outil. L'exemple dans cette figure expose la spécification d'un outil nommé Chat. Cet outil est lié à une applet ChatApplet qui est un client Chat (IRC ou babillard). Les opérations de cet outil permettent de Demarrer l'outil, et de Bloquer ou Debloquer le bouton qui permet l'envoi de message sur le canal de conversation auquel AppletChat est connectée lorsqu'elle est instanciée.

La Figure 43 est un exemple de boîte de dialogue permettant aux sujets de créer ou redéfinir une opération particulière. Cette boîte apparaît lorsqu'un sujet demande à l'AppletOutil une création ou une édition d'opération pour l'outil. Rappelons que chaque opération est un script constitué d'une séquence de méthodes à invoquer sur l'applet liée. La boîte de dialogue permet de donner le nom de l'opération, sa description, et enfin de construire le script à exécuter lors du déclenchement de l'opération. Dans notre exemple de boîte de dialogue, le

sujet voit dans la partie droite du cadre *Methods* la liste des méthodes récupérées sur l'applet liée. La partie gauche de ce cadre représente les méthodes qui seront appelées par l'opération définie. Le sujet utilise les deux boutons centraux pour ajouter ou retirer des méthodes de son script. Lors de l'exécution de l'opération, les méthodes de la partie gauche seront exécutées en séquence. Ni ce mode séquentiel, ni le fait que les paramètres des méthodes ne soient pas pris en compte, ne sont directement liés à l'implémentation de DARE. Etant donné que chaque opération est traduite en une méthode Smalltalk, il serait tout à fait possible de créer des scripts complexes utilisant les formalismes et structures de contrôle classiques des langages de programmation. Toutefois, rappelons que ce sont les sujets qui définissent les opérations. Cet exemple démontre déjà la complexité d'une telle définition, ce qui explique nos choix quant à ces limitations. Lorsque la définition est validée par le sujet, une méthode d'instance de la classe représentant l'outil est créée sur le serveur d'activités.

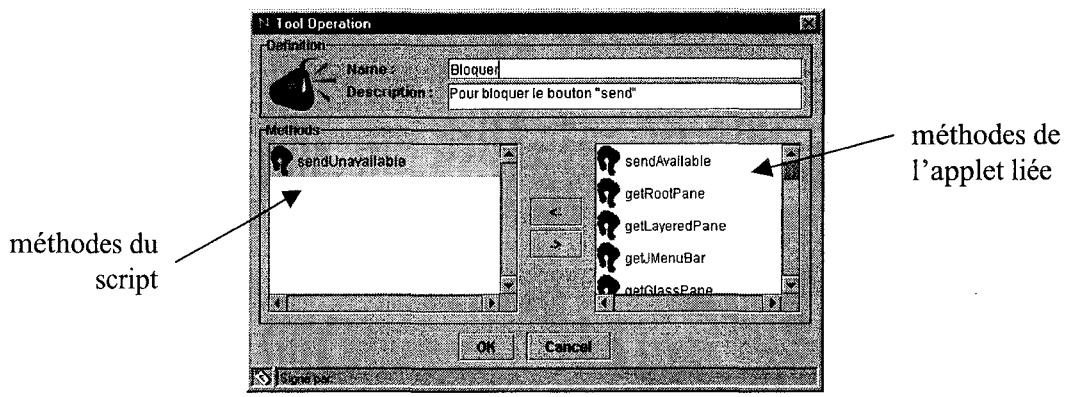

Figure 43. La création d'opération (pour l'outil Chat de la Figure 42)

D'une manière plus précise, l'exemple de la Figure 43 porte sur l'outil Chat introduit précédemment et lié à l'applet ChatApplet. Nous nous plaçons dans l'hypothèse de la création d'un schéma de coopération où Chat sera utilisé dans un *mode conférence*. Dans ce mode de coopération, un *conférencier* peut envoyer des messages sur le canal de communication de ChatApplet, alors que les *auditeurs* ont leur bouton d'envoi bloqué. Une des étapes dans la création de ce mode de coopération au sein de DARE consiste à spécifier une opération permettant de bloquer le bouton d'envoi de message de l'applet liée. Notons que ceci n'est réalisable que si l'applet ChatApplet possède déjà une ou des méthodes permettant de bloquer ce bouton. Le sujet définit donc une opération Bloquer dans Chat. Pour ce faire, il sélectionne la méthode sendUnavailable de ChatApplet et l'ajoute à son script (qui ne contient ici qu'une méthode). Lorsque l'opération est validée, il est possible de déclencher Bloquer sur toute instance de l'outil Chat. Notons que cette création d'opération peut être réalisée alors que des

instances de l'outil Chat existent déjà dans des supports d'activité, ceci grâce aux mécanismes réflexifs de DARE.

Cet exemple démontre la difficulté introduite par la création d'un lien fin entre une applet extérieure à DARE et un outil. Dans la Figure 43, focalisons sur les méthodes de ChatApplet utilisables pour créer des opérations dans Chat (les *méthodes de l'applet liée*). Les mécanismes d'introspection de Java nous permettent de récupérer la liste des méthodes de l'applet. Cependant, cette liste s'avère peu compréhensible, ce qui augmente la difficulté du processus de création d'opération, et donc du processus de liaison en général. Il serait possible de filtrer l'ensemble de ces méthodes pour ne proposer au sujet que celles accessibles à son niveau d'abstraction, en utilisant des conventions similaires à celles des JavaBeans par exemple. Mais, pour pouvoir filtrer, il faut que le développeur de l'applet ait introduit une convention de filtrage et que DARE en connaisse les spécifications ! Ce n'est actuellement pas le cas.

7.4.3 Utiliser une applet liée à DARE

L'utilisation d'une applet liée à DARE correspond à utiliser un des outils spécifiés dans la tâche du support d'activité. Concrètement, un sujet démarre une applet à partir de son applet activité. Cette applet est alors manipulée par le sujet et 'pilotée' par le serveur d'activités. Ce pilotage correspond aux répercussions des actions des autres sujets. Les actions qui peuvent être déclenchées au niveau du support d'activité sont celles spécifiées dans les micro-rôles. Chaque action exécute un script d'opérations sur l'outil. Chaque opération doit invoquer un ensemble de méthodes sur l'applet liée. Il faut donc créer une connexion entre l'outil du serveur d'activités et l'applet liée instanciée par le sujet dans son navigateur.

De tels mécanismes sont présentés dans la Figure 44. Un outil instancié (dans la collection d'outils d'un membre) génère un document HTML à partir d'un modèle générique. Ce document contient l'URL de l'applet liée (UneApplet) ainsi que l'URL d'une applet invisible de DARE, nommé AppletControl et permettant de contrôler l'exécution de l'applet liée. AppletControl a pour paramètre une référence d'objet CORBA lié à l'outil du membre. Ainsi, dès son démarrage, AppletControl se connecte au serveur d'activités. Lorsqu'une action est déclenchée sur le micro-rôle d'un membre (1) (par le membre lui-même ou par le support d'activité), elle invoque une série d'opérations sur l'outil. Chaque opération se traduit en invocations de méthodes transmises à AppletControl (2). AppletControl, ayant dès son démarrage

récupéré une référence sur UneApplet, invoque alors ces méthodes sur sa cible. Ceci permet de piloter à partir de DARE et par l'ORB une applet à priori extérieure et non-CORBA.

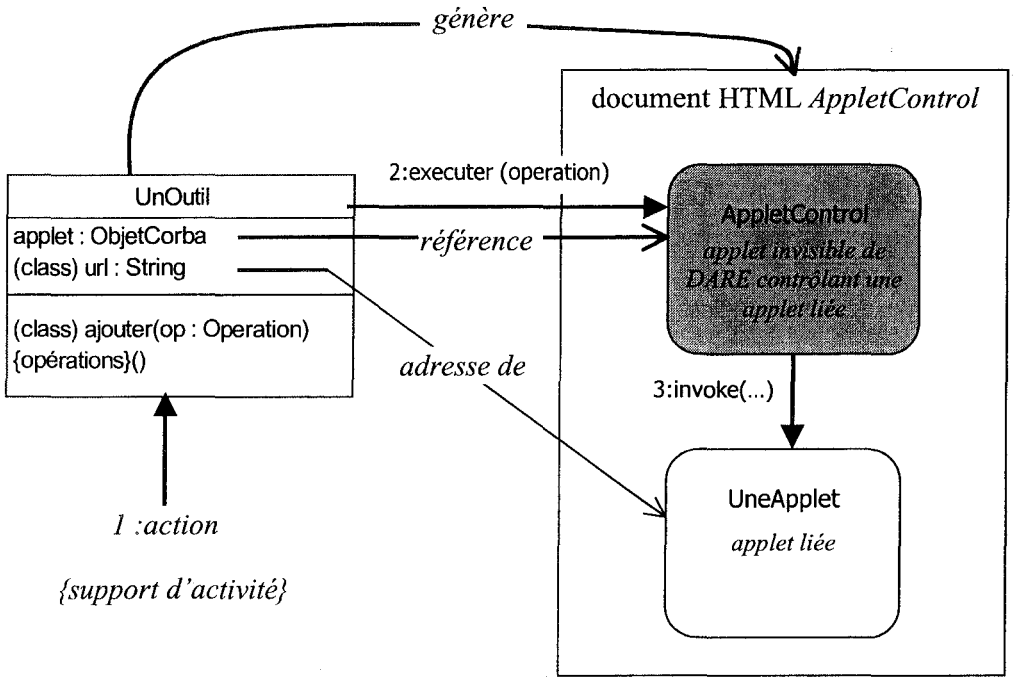

Figure 44. Le contrôle d'une applet liée

Grâce à ce modèle architectural, il est aussi techniquement possible de recevoir des événements (comme les déplacements de la souris) ayant été générés par l'applet liée elle-même. En effet, java.applet.Applet étant une sous-classe de java.awt.Component, AppletControl peut s'enregistrer dans les services d'événements de UneApplet. La Figure 45 montre comment, une fois enregistrée (1), AppletControl reçoit et transmet des événements vers son outil, qui les retransmet au support d'activité. Ces événements, s'ils sont traduits dans le contexte de la tâche, peuvent avoir des répercussions dans tout le support d'activité (d'une manière similaire aux actions des micro-rôles).

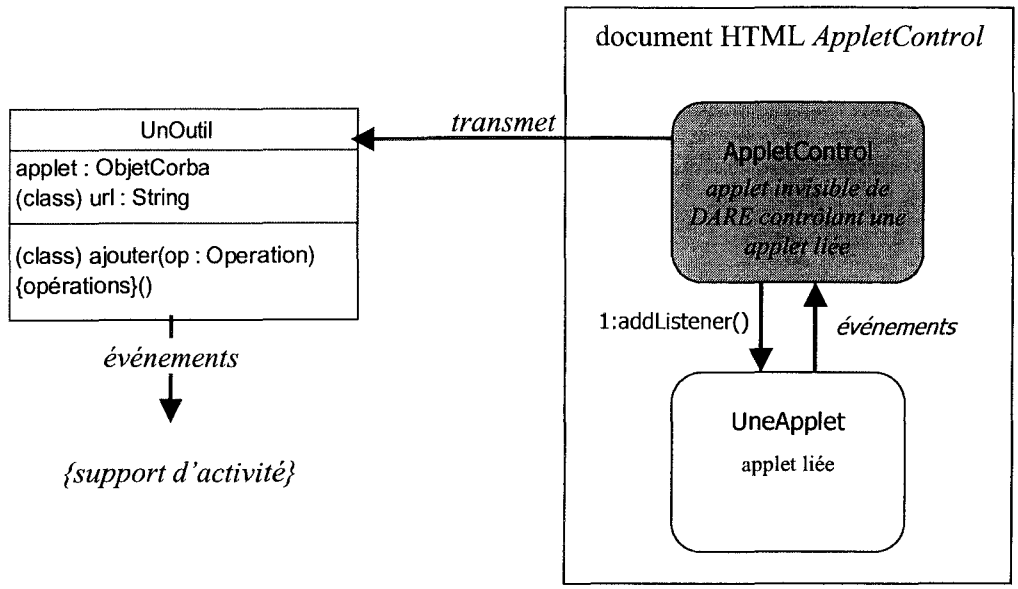

Figure 45. La répercussion d'événements générés dans une applet liée

L'architecture de DARE rend donc techniquement possible la récupération des actions directes des sujets sur les applets outils. Il suffit de leur donner un sens dans le support d'activité. Malheureusement, la mise en place de tels mécanismes est encore une fois limitée par la réalisation des applets liées. En effet, pour traduire une action directe d'un sujet sur un outil en action d'un micro-rôle, il faut pouvoir (1) recevoir un événement et (2) comprendre cet événement. Le point (1), même si comme nous avons montré qu'il est réalisable, dépend totalement de l'implémentation de l'applet liée. Une applet étant constituée d'une structure hiérarchique de composants (les canevas, les boutons, les listes, etc.), il est difficile de savoir quel composant écouter pour récupérer un événement intéressant. Le point (2) est plus subjectif dans le sens où un événement n'est souvent compréhensible qu'à un niveau d'abstraction très proche du composant qui l'a généré. L'idéal serait que l'applet elle-même propose une interface méta permettant un enregistrement dans des services d'événements bien identifiés et compréhensibles par les sujets, c'est-à-dire, à un niveau d'abstraction proche de la tâche. Une fois encore, ce type d'applets est pour l'instant[1] difficile à trouver. C'est pourquoi, dans cette première version de DARE, nous avons opté pour un déclenchement délocalisé des actions des micro-rôles. Ce déclenchement est effectué à partir de l'applet activité.

[1] Nous travaillons actuellement sur un modèle de composant applet adapté à nos besoins

7.4.4 Les actions des micro-rôles

Une action d'un micro-rôle peut être *visible* ou *invisible* et *locale* ou *globale*. La Figure 46 donne un exemple de déclenchement d'actions sur un outil dans l'applet activité d'un membre. Les actions disponibles pour le membre sur un outil sont obtenues en fonction de ses micro-rôles dans un menu contextuel (click droit). Les actions proposées ne sont que les actions visibles des micro-rôles du membre. Dans notre exemple d'interface basique, les actions locales et globales sont mélangées. L'action Demarrer proposée sur l'outil Chat est un exemple d'action locale. Les actions Conference et Anarchique sont des actions globales.

Une *action locale* ne déclenche des opérations que sur l'outil du membre qui l'exécute. Autrement dit, le membre n'agit que sur son propre outil et n'influence pas directement ceux des autres. Par exemple, lorsque l'action Demarrer est déclenchée, l'appel est transmis à l'objet représentant le micro-rôle sollicité sur le serveur, qui déclenche à son tour les opérations sur l'outil. En réponse à l'action démarrer, l'applet activité reçoit les informations (url, etc.) lui permettant de démarrer l'applet ChatApplet liée. Cette action locale n'a pas d'effet direct sur l'outil Chat des autres membres du support d'activité. En d'autres termes, les outils Chat des autres membres ne sont pas automatiquement démarrés.

Une action globale est déclenchée sur tout micro-rôle du support d'activité qui est lié à l'outil visé et qui l'implémente. Ceci introduit une contrainte : un membre ne doit pas posséder plusieurs micro-rôles qui sont liés à un même outil et qui définissent des actions du même nom. Cette contrainte est d'autant plus importante que le script de l'action déclenchée varie généralement d'un micro-rôle à l'autre. Prenons un exemple...

Le membre qui voit l'applet activité de la Figure 46 joue le rôle d'Enseignant. Ce rôle lui donne un ensemble de micro-rôles. L'un d'entre eux est le micro-rôle Conferencier qui est associé à l'outil Chat vu précédemment. Ce micro-rôle définit et donne accès aux actions Anarchique, Demarrer et Conference. Demarrer permet au membre de démarrer son outil Chat. Anarchique permet de passer dans un mode de coopération dans lequel tous les membres du support d'activité peuvent 'parler' en même temps. Conference correspond au mode de coopération décrit précédemment et dans lequel seul le Conferencier a le droit de 'parler'. Ces trois actions sont visibles pour ce micro-rôle. Les autres membres du support d'activité jouent le rôle d'Etudiant. Chaque étudiant possède le micro-rôle Auditeur défini pour l'outil Chat. Ce micro-rôle définit lui aussi les actions Anarchique, Demarrer et Conference. La différence est que pour l'Auditeur, la seule action visible est Demarrer. De ce fait, un Etudiant n'aura accès qu'à

cette action par un click droit sur l'outil Chat dans son applet activité. Les actions Conference et Anarchique de Conferencier sont globales, alors que celles du même nom pour l'Auditeur sont locales. Ainsi par exemple, le déclenchement de Conference par le Conferencier a pour effet le déclenchement de Conference sur tous les Auditeur et Conferencier du support d'activité. Il est à noter qu'alors, l'action Conference des autres Conferencier est exécutée comme une action locale du micro-rôle, ceci pour éviter un appel récursif.

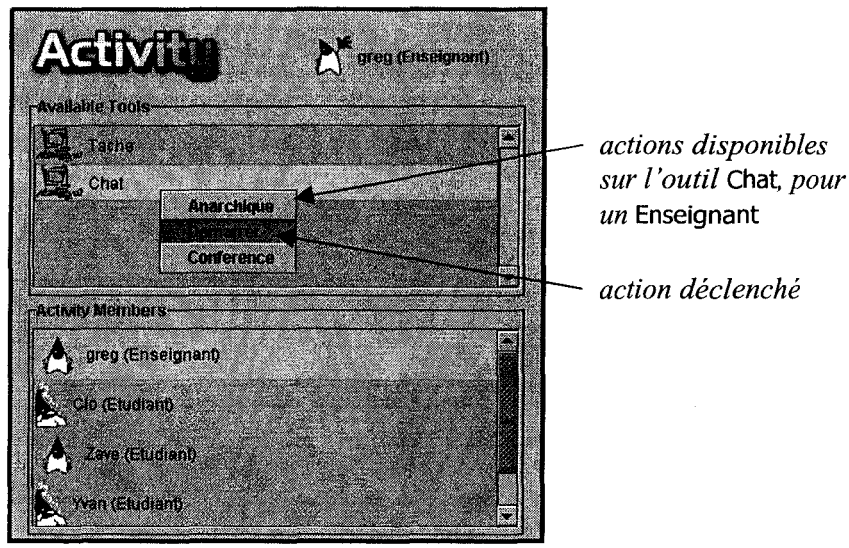

Figure 46. Un déclenchement d'action sur l'outil Chat à partir de l'applet activité

Nous avons vu précédemment la création d'une opération Bloquer sur l'outil Chat et ayant pour effet de bloquer le bouton d'envoi de message de l'applet liée. L'outil Chat possède aussi une opération Debloquer qui a l'effet inverse. L'action Conference de Conferencier déclenche l'opération Debloquer, alors que l'action du même nom d'Auditeur déclenche l'action Bloquer. L'action Conference déclenchée par un Enseignant (donc Conferencier) a pour effet de modifier le mode d'utilisation des instances de l'applet ChatApplet liée. La Figure 47 fournit une représentation de l'état de l'applet ChatApplet suite au déclenchement de Conference par un Conferencier : l'applet liée d'un Conferencier permet d'envoyer des messages sur le canal de conversation, celle d'un Auditeur ne le permet plus.

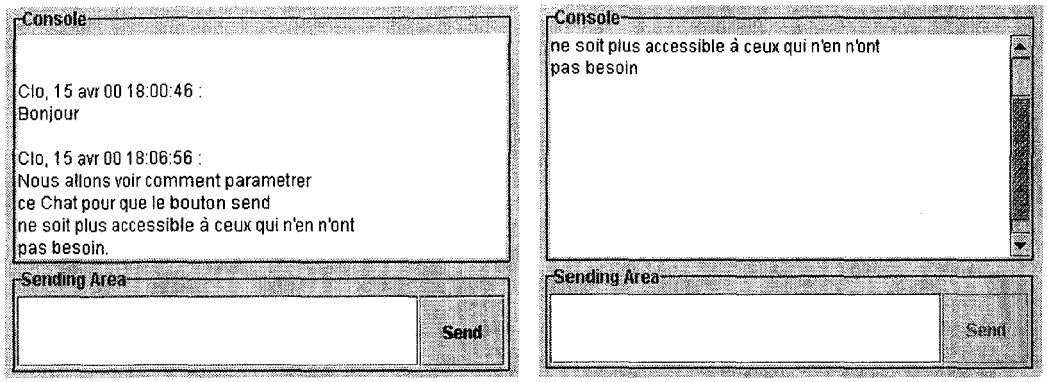

Figure 47. L'applet ChatApplet des différents membres suite à l'action Conference d'un Enseignant (Conferencier)

Nous savons donc lier une applet extérieure pour en faire un outil des sujets. Toutefois, au delà des problèmes techniques rencontrés, subsistent d'autres problèmes : quels types d'outils (au niveau sémantique) pouvons-nous intégrer dans DARE ? Nous identifions trois principaux types d'outils informatiques pouvant être intégrés par les sujets: les *outils coopératifs*, les *espaces partagés* et les *outils isolés*.

7.4.5 Les applets coopératives

Les applets coopératives sont à priori les plus proches de notre domaine, ce qui devrait faciliter leur intégration dans DARE. Une applet/outil de vote ou encore un éditeur de texte coopératif en sont des exemples.

Les caractéristiques d'une applet coopérative sont de définir à la fois un espace partagé ainsi qu'un espace de coopération. Le fait qu'une applet coopérative gère ses propres règles de coopération est à la fois un atout et un problème dans le cadre de DARE.

Les applets gérant leurs propres règles de coopération facilitent leur intégration. Ces applets utilisent généralement des notions proches de celle du micro-rôle de DARE (des rôles associés à un seul outil). Il suffit de spécifier la projection des rôles de l'applet coopérative vers les micro-rôles de DARE. Prenons par exemple le cas d'une applet/outil de vote. Si cette applet définit elle-même les rôles *initiateur* du vote ou *votant*, il est possible de définir dans DARE des micro-rôles similaires sémantiquement. Lorsqu'un sujet démarre son outil de vote, le micro-rôle (de DARE) qu'il joue dans le support d'activité configure automatiquement

l'applet coopérative liée (le client instancié dans le navigateur) pour ce sujet en déclenchant par exemple une méthode initialiser('votant').

Malheureusement, les principes de base de DARE spécifient qu'une communauté doit être capable de redéfinir les rôles en cours d'activité. Nous avons d'ailleurs spécifié une architecture cadre basée sur l'implémentation ouverte pour aller dans ce sens. Le problème d'une applet liée gérant sa propre coopération est qu'elle n'est généralement pas aussi souple que le noyau de DARE. S'il est impossible de redéfinir les règles de coopérations de l'applet coopérative liée à DARE, il n'est pas possible de les redéfinir, pour l'outil correspondant, dans DARE.

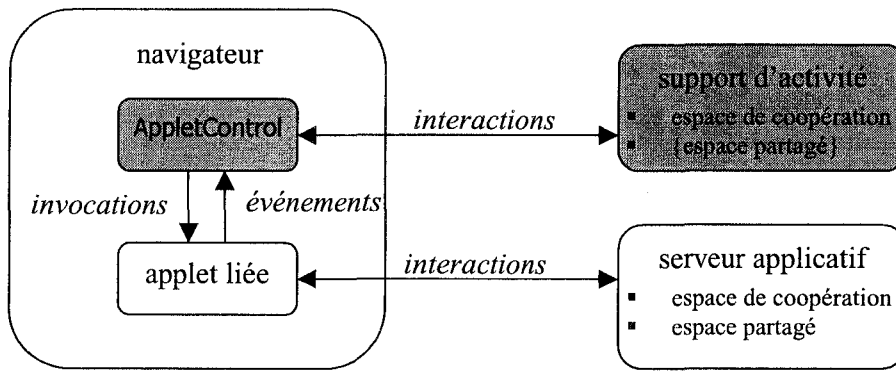

Figure 48. Utilisation d'une applet coopérative liée à DARE

Comme nous l'avons souligné, une applet coopérative gère son propre espace partagé. C'est pourquoi il est peu probable que DARE gère lui-même un espace partagé pour l'outil correspondant. Toutefois, grâce aux mécanismes de récupération d'événements spécifiés précédemment, il est possible (en fonction de l'ouverture de l'applet coopérative) d'étendre cet espace partagé. La Figure 48 résume ces différents points.

7.4.6 Les espaces partagés ou applets multi-utilisateurs

L'exemple typique d'applet gérant simplement un espace partagé est le 'chat' (aussi appelé IRC ou encore babillard). Un autre exemple est une simple[1] applet d'audio conférence. L'intégration fine (mettant en œuvre des règles de médiation) dans un support d'activité implique ici la création d'un espace de coopération entièrement géré par DARE. A la

[1] sans gestion de tour de parole, etc.

différence d'une applet coopérative, il n'y a pas risque de conflits entre les règles de coopération du support d'activité et celles de l'applet liée (qui n'existent pas). Les micro-rôles peuvent être totalement redéfinis si besoin. Malheureusement, les règles doivent être totalement spécifiées, ce qui implique un surcroît de travail non trivial. Une fois encore une extension de l'espace partagé peut être envisagée (si l'applet le permet). La Figure 49 résume ces différents points.

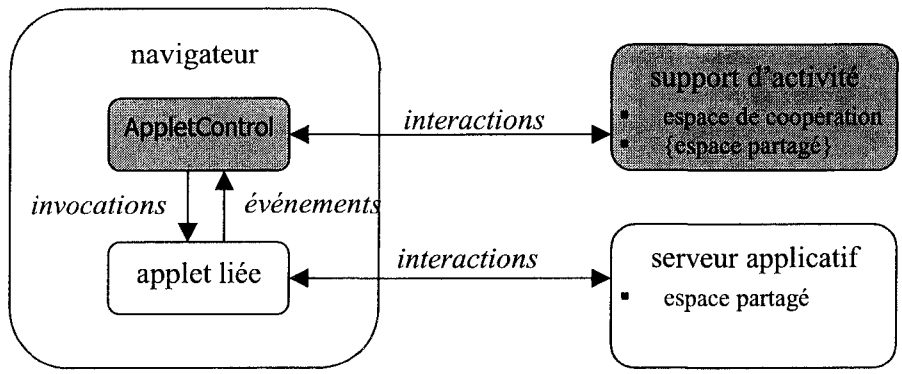

Figure 49. Utilisation d'une applet partagée liée à DARE

7.4.7 Les applets isolées

Les applets isolées ne gèrent ni espace partagé, ni espace de coopération. Grâce à son modèle architectural, DARE est techniquement capable de faire d'une applet isolé une applet partagée ou coopérative (cf. Figure 50). Ceci signifie le routage d'événements générés dans l'applet d'un membre vers les applets correspondantes des autres membres, ainsi que la création d'un espace de coopération (micro-rôles). Le point positif, dans ce cas est que pratiquement tout peut être redéfini au cours de l'activité. Le point négatif est que tout est à faire ! Notons toutefois qu'une applet isolée peut parfaitement être utilisée en tant que telle. Par exemple, lier une applet calculette à DARE, sans créer d'espace partagé ni de règles de coopération, permet à chaque sujet d'avoir à disposition une calculette individuelle dans son support d'activité.

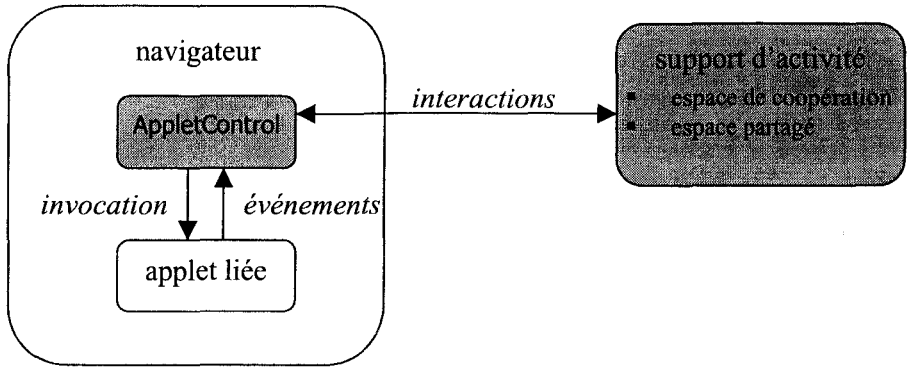

Figure 50. Utilisation d'une applet isolée liée à DARE

7.4.8 La malléabilité dans l'outil

Le lien créé entre l'outil et son applet liée est un lien malléable. Il peut être redéfini par les sujets, dans leur support d'activité, et au cours de l'utilisation de l'outil lui-même. Cette malléabilité permet l'intégration : une applet liée est une applet intégrée dans DARE par le biais de l'outil. Elle permet aussi l'extension : l'ajout, la modification ou le retrait d'opérations dans un outil est un exemple typique d'extension ou de restriction de ses capacités.

Une certaine malléabilité peut aussi exister dans l'applet liée à l'outil. Toutefois, cette malléabilité dépend totalement des choix de ses développeurs. L'applet peut être paramétrable. Elle peut elle-même être faite de composants et permettre une certaine intégration. L'extension au niveau de l'applet est peu probable étant donné les limites de Java à ce niveau. Cependant, si cette applet est la partie cliente d'un système coopératif, il est possible (comme dans DARE) qu'une partie de ce système soit réalisée dans un langage permettant l'extension.

Le contrôle de DARE sur cette malléabilité dépend totalement, une fois encore, de l'ouverture de l'applet, c'est-à-dire des méthodes pouvant être récupérées par introspection. L'utilisation de la méthode sendUnavailable() pour créer l'opération Bloquer dans notre exemple du Chat est un exemple de paramétrage d'une applet à partir de DARE.

7.5 Les outils et applets méta

7.5.1 Quels sont-ils ?

Les outils méta sont principalement ceux qui ont été développés pour DARE et dans le cadre de cette thèse. Une fois encore, nous voulons insister sur le fait que cette première réalisation des outils méta a principalement été motivée par la volonté de valider notre framework. Nous sommes conscients que ces outils demandent encore un gros travail en terme d'interface homme-machine. Nous reviendrons d'ailleurs sur ce sujet dans le chapitre 8.

Les outils méta permettent aux sujets d'accéder au niveau méta de leur support d'activité. Ils permettent de spécifier des tâches, des outils, des rôles et des micro-rôles et de créer les liens entre ces composants. Nous avons déjà rencontré certains de ces outils. L'applet AppletOutil utilisée dans les paragraphes précédents constitue la partie cliente d'un outil méta utilisé pour accéder à la définition des outils d'une tâche. Plus généralement, *tout outil utilisant l'interface méta de DARE pour accéder en introspection et/ou intercessions aux modèles d'activités du serveur d'activités est un outil méta.*

7.5.2 Des outils comme les autres

De notre point de vue, l'intérêt principal de DARE provient du fait que les outils méta sont des outils à part entière. Ils sont accessibles au sein des supports d'activité qu'ils servent à modifier. De ce fait, tout support d'activité est normalement redéfini par ses sujets (membres) et au cours de sa propre exécution.

Comme les autres outils, les outils méta sont liés à une applet. Ils sont des outils coopératifs. L'espace partagé contient les composants qui constituent le support d'activité dans lequel ils sont eux-mêmes instanciés. Les règles de coopération sont gérées par le serveur d'activités et spécifiées dans la tâche qui les utilise. Ainsi, un support d'activité est aussi un support de méta-activité dans lequel les outils méta peuvent être utilisés pour s'auto-spécifier.

7.5.3 Un exemple d'utilisation

La Figure 51 expose l'approche réflexive que nous avons adoptée pour la définition et l'utilisation des outils méta. Cet exemple expose comment, dans DARE, le niveau de

l'activité et de la méta-activité des sujets sont fusionnées : *DARE se considère lui-même comme un outil à part entière.*

Pour définir ajouter ou modifier un outil dans un support d'activité, il faut accéder à la tâche de ce support. L'outil Tache permettant de redéfinir la tâche d'un support d'activité est accessible via l'applet activité liée à ce support. Notons toutefois que l'outil Tache n'apparaît dans l'applet activité d'un sujet que si celui-ci joue un rôle (i.e. des micro-rôles) lui permettant d'y accéder. Si c'est le cas, le sujet possède une collection d'actions définies dans ses micro-rôles lui permettant d'agir sur cet outil. En particulier, l'une de ces actions permet certainement de démarrer l'applet AppletTache liée à l'outil Tache de ce support d'activité.

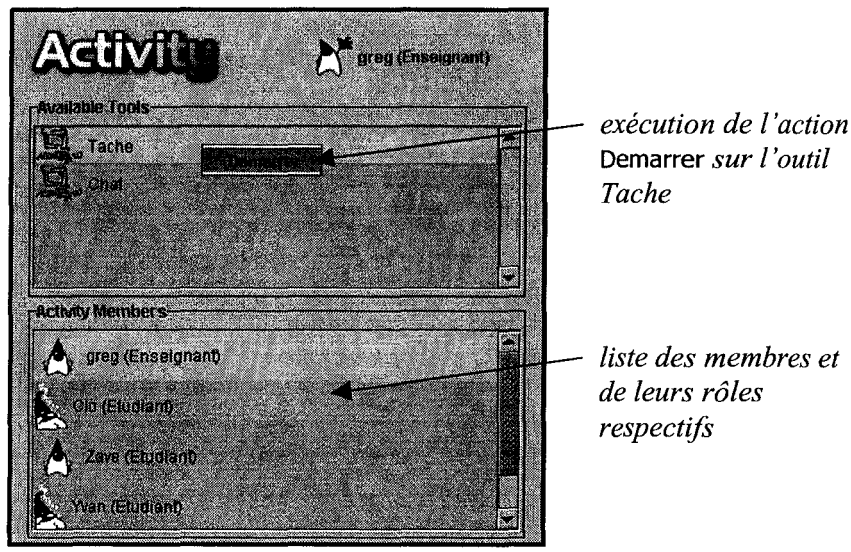

Figure 51. L'accès à la tâche au sein du support d'activité

Le démarrage d'AppletTache permet au sujet d'accéder à la spécification de la tâche en cours de réalisation. La Figure 52 représente cette applet telle qu'elle est actuellement réalisée. On pourra remarquer que l'objet de l'activité y est spécifié. Cette applet présente aussi la liste des outils et rôles spécifiés pour cette tâche. Pour ce faire, AppletTache contient une référence d'objet CORBA directement lié à la classe (sous-classe de Task) du support d'activité en cours d'exécution. Notons que dans cette version, AppletTache ne tient pas compte des sous-tâche qui peuvent exister dans la tâche en cours d'édition. Elle n'utilise donc pas toutes les possibilités de l'architecture cadre de DARE.

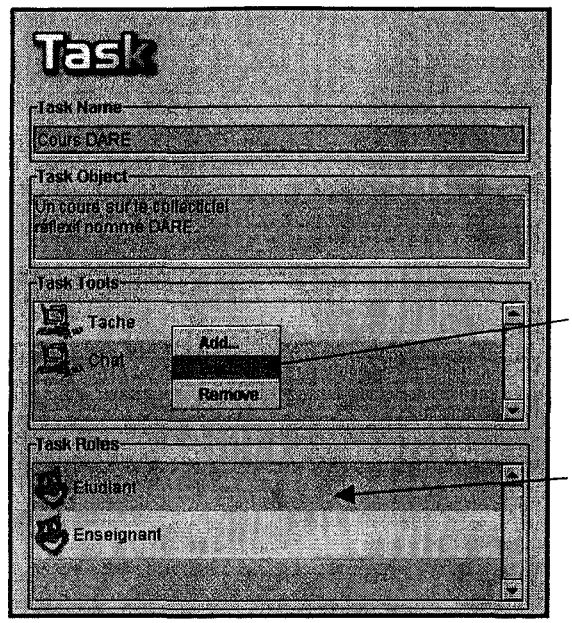

Figure 52. L'applet de spécification de la tâche : AppletTache

Les informations présentées sont obtenues grâce à l'interface méta du composant tâche. Cette interface méta définit aussi les moyens pour les modifications du modèle (patron) d'activité en cours de spécification. Il existe donc un fichier IDL qui spécifie les opérations de l'interface méta du composant Task. Ceci est vrai pour tout type de composant défini dans le méta-modèle de DARE. Ainsi, chaque outil méta destiné à manipuler un type de composant de DARE (Task, Tool, etc.), utilise les services définis dans une interface IDL proche de l'interface méta de ce type de composant. La Figure 53 donne l'interface IDL de l'interface méta des composants qui héritent de Task.

```
typedef sequence<OutilClass> TableauOutilClass ;
typedef sequence<RoleClass> TableauRoleClass ;
typedef sequence<TacheClass> TableauTacheClass ;

interface TacheClass{
 string recupererObject() ;
 void fixerObject(in string object) ;
 TableauOutilClass listerOutils() ;
 TableauRoleClass listerRoles() ;
 TableauTacheClass listerSousTaches() ;
 void ajouterOutil(in OutilClass outil) ;
 void retirerOutil(in OutilClass outil) ; ... };
```

Figure 53. Interface IDL de l'interface méta d'une tâche

AppletTache permet, entre autre, d'ajouter, de retirer ou de modifier les outils de la tâche. Dans la Figure 52, le sujet décide d'éditer l'outil Tache de la tâche. On voit alors apparaître toute la réflexivité de DARE : l'outil Tache lié à AppletTache est utilisé pour se redéfinir lui-même.

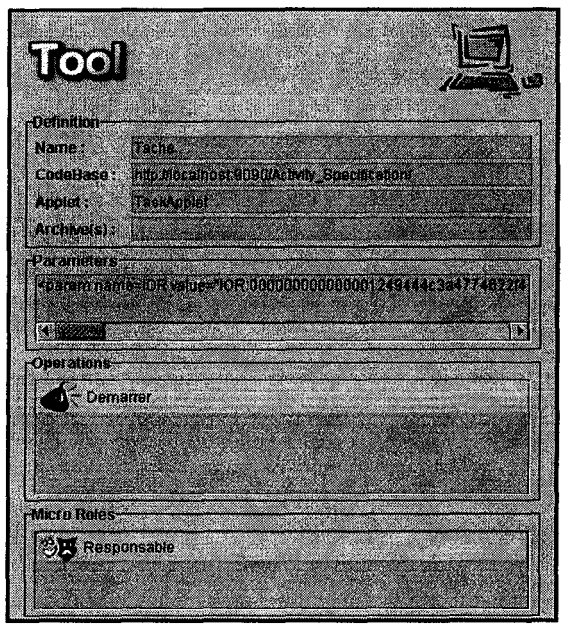

Figure 54. Définition du lien sur l'outil d'accès à la tâche

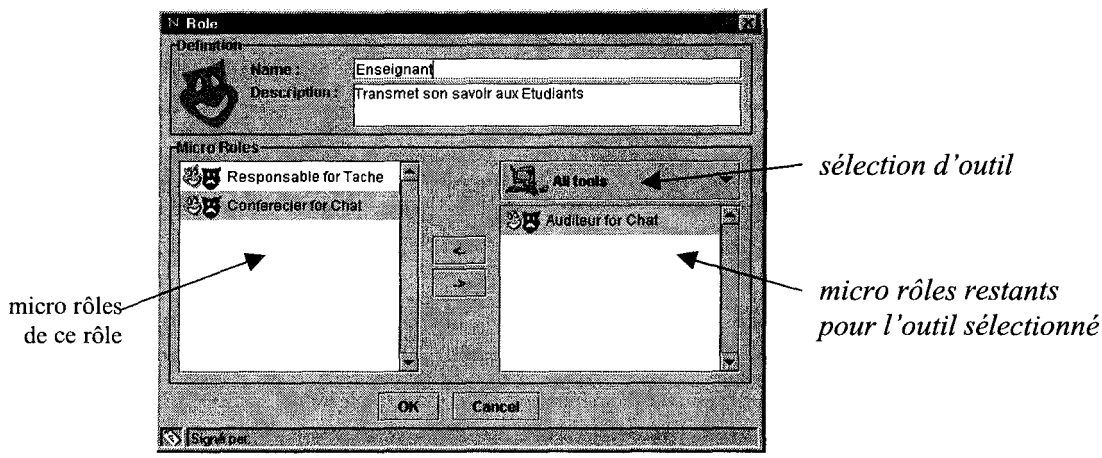

Figure 55. Spécification d'un rôle et d'un micro-rôle

La demande d'édition a pour conséquence d'instancier une AppletOutil sur l'outil Tache (cf. Figure 54). Dans notre exemple, on voit entre autre apparaître l'opération Demarrer de cet outil, qui a servi à spécifier l'action Demarrer du micro-rôle Responsable faisant parti du rôle Enseignant détenu par le sujet. C'est ce micro-rôle qui a permis au sujet d'accéder au niveau

méta de son support d'activité (cf. Figure 51). Ce point peut être vérifié ou modifié à partir d'AppletTache en éditant le rôle Enseignant (cf. Figure 55).

7.6 Conclusion

Ce chapitre montre la puissance d'intégration et la malléabilité de DARE pour créer des environnements de TCAO. Ces environnements peuvent être (re)spécifiés par leurs propres utilisateurs. DARE permet de spécifier des supports d'activité dont chaque composant est malléable, d'intégrer des systèmes informatiques externes, et de redéfinir sa propre utilisation au sein de chaque support d'activité. En résumé :

- DARE est un environnement de TCAO :
  - global
  - intégrateur
  - malléable
  - supportant la mobilité des sujets
- DARE se considère comme un outil à part entière.
- DARE supporte sa propre activité coopérative de (re)spécification.

Tableau 8. Les propriétés de DARE du point de vue du sujet

Toutefois, la vision optimiste Tableau 8 est quelque peu atténuée devant les problèmes posés pour la mise en pratique des propriétés précitées. En particulier, nous avons insisté sur la complexité introduite principalement dans le processus de liaison des applets. Nous avons fait le choix de l'ouverture maximale en proposant de pouvoir intégrer plus ou moins finement n'importe quelle applet découverte sur Internet. Les parties précédentes montrent que cette démarche est techniquement réalisable (puisque réalisée). Le problème est que les applets sont conçues pour être intégrées dans un environnement d'exécution comme un navigateur, mais pas dans un environnement de TCAO. Ainsi, les méthodes qu'elles proposent sont plus tournées vers les moyens de leur intégration au niveau logiciel, que vers le niveau d'abstraction des sujets qui les utilisent. De notre point de vue, ceci engendre deux problèmes majeurs pour l'utilisation d'applets récupérées sur le réseau (qu'elles soient isolées, partagées ou coopératives) : soit ces applets ne proposent pas de méthodes utiles au niveau d'abstraction des sujets ; soit ces méthodes sont très difficilement identifiables (elles sont mélangées aux

autres méthodes). D'une manière générale, les méthodes de l'applet sont difficilement compréhensibles et donc peu utilisables : il n'y pas de « manuel utilisateur » des méthodes de l'applet.

Ce chapitre conclue la présentation de notre réalisation dans le cadre de cette thèse. Il nous reste maintenant à dresser le bilan de nos travaux ainsi que les premiers enseignements que nous avons pu tirer de cette réalisation et de ses résultats.

Chapitre 8

Bilan des travaux et perspectives...

Ce chapitre présente le bilan de nos travaux. Il évalue dans un premier temps la satisfaction des enjeux que nous nous sommes fixés suite à notre étude de la Théorie de l'Activité et dans la proposition de nouveaux fondements pour la création d'environnements de TCAO. Nous étudions ensuite l'impact d'un tel environnement, défini comme un *collecticiel réflexif*, sur le processus de conception des environnements de TCAO. Nous présentons le bilan des moyens que nous avons mis en œuvre pour réaliser DARE en insistant sur les problèmes toujours existants. Nous comparons nos propres travaux aux travaux analogues et touchant les principaux thèmes que nous avons abordés. Enfin, nous terminons par la présentation d'un certain nombre de nouveaux travaux déjà entamés, à partir de nos résultats de recherche, et suite à la réalisation de DARE.

8.1 La satisfaction des enjeux

Les enjeux que nous nous sommes fixés dans cette thèse (chapitre3) suite à notre étude de la Théorie de l'Activité sont les suivants :

1) Offrir un support à l'activité selon les besoins exprimés par les sujets.
2) Supporter la redéfinition du support à l'activité au cours de l'activité.
3) Cristalliser l'expérience des sujets sous forme de patterns d'activités.
4) Permettre la réutilisation de patterns pour démarrer une nouvelle activité.
5) L'évolution d'une activité doit être un processus coopératif réalisé in situ.

Nous pensons avoir atteint ces objectifs, tout au moins dans leurs lignes générales. Néanmoins, nous souhaitons souligner qu'il ne s'agit ici que de l'évaluation du potentiel de notre réalisation. En effet, chaque enjeu ne pourra être déclaré complètement atteint que lorsque l'ensemble des travaux qui suivent et se basent sur notre réalisation auront été menés à bien. En particulier, l'absence d'Interfaces Homme-Machine adaptées dans la version actuelle de DARE ne permet pas d'étudier la bonne utilisation des propriétés fondamentales de DARE par ses utilisateurs.

L'enjeu 1) est réalisé dans la mesure où DARE permet effectivement aux sujets d'exprimer leurs besoins envers leur support d'activité. L'expression de ces besoins est basée sur le modèle d'Engeström de *la structure basique de l'activité.* Ce modèle nous a permis de construire un *modèle générique* de support d'activité ainsi qu'un *méta-modèle* permettant de manipuler tout modèle de *support d'activité.* Du point de vue des sujets, les spécifications sont réalisées au travers d'applets Java que nous avons appelées *applets méta.*

L'enjeu 2) est lui aussi atteint. Un support d'activité est redéfini au cours de sa propre exécution. Cette propriété a été réalisée grâce à l'introduction d'un *modèle réflexif.* Le niveau méta d'un support d'activité est contenu et accessible dans ce dernier. Dans le souci de lier finement les fondements de DARE à l'AT, le niveau méta a été identifié comme la *tâche.* Il existe une relation causale entre une tâche et ses supports d'activité. Cette réalisation a été facilitée en utilisant les mécanismes réflexifs des langages de programmation utilisés.

L'enjeu 3) a été atteint. DARE permet de *cristalliser l'expérience* des sujets au sein des composants qu'ils définissent, manipulent et modifient. Ces composants sont de plusieurs types. Il s'agit de composants Tache, Outil, Role et MicroRole. Dans DARE, chaque composant correspond à une classe et possède des mécanismes réflexifs qui permettent de le comprendre, de l'utiliser et de le modifier. Les classes évoluent de par leur manipulation par les sujets. Elles cristallisent ainsi leur expérience de la situation. En particulier, un composant Tache contient en lui toutes les informations nécessaires à la création d'un support d'activité. De ce point de vue, une Tache est un patron d'activité qui peut être instancié et spécialisé.

Nous considérons l'enjeu 4) comme partiellement atteint. Le noyau de DARE permet bien d'instancier un patron d'activité (ou composant Tache) un nombre quelconque de fois. Il est aussi possible d'hériter d'une Tache ou d'un de ses composants, pour en créer de nouveaux du même type. Ces nouveaux composants peuvent alors être eux-mêmes spécialisés. Ils cristallisent alors une nouvelle expérience. Tous ces mécanismes sont implémentés au sein de

DARE. Toutefois, notre réalisation au niveau des applets méta n'a pas réellement pris en compte cette dimension. En fait, la partie client de DARE n'utilise pas encore toute la puissance du noyau. Par exemple, les applets méta ne permettent pas encore d'accéder au dépôt de composants disponibles dans le serveur. Cependant, rappelons que ces applets sont destinées à évoluer et que, selon l'architecture de DARE, il sera simple de les remplacer par de nouvelles versions. En particulier, les applets méta existantes pourront être utilisées pour se remplacer elles-mêmes. En effet, il suffira de modifier le lien entre l'outil méta correspondant et son applet liée : la nouvelle applet méta remplace l'ancienne.

L'enjeu 5) est atteint. Lors de la présentation du modèle réflexif et architectural de DARE, nous avons vu que les outils méta sont des outils à part entière du support d'activité. Il est en particulier possible de définir ou de redéfinir les règles de coopération qui s'y rapportent. Ainsi, le processus d'évolution d'une activité est bien un processus coopératif réalisé in situ. Autrement dit, dans DARE, le support d'activité coopérative et son support de méta-activité sont fusionnés.

Nous pensons donc avoir globalement atteint les objectifs que nous nous étions fixés dans le but de créer un nouveau genre d'environnement pour le TCAO : un collecticiel intimement lié aux fondements de l'activité humaine. Il nous faut maintenant étudier l'impact de l'introduction d'un tel environnement pour le TCAO.

8.2 Impact sur le processus de conception

Les travaux réalisés dans le cadre de cette thèse ne sont que les bases d'un nouveau genre de collecticiel. Ces bases, du point de vue de l'utilisateur, ne peuvent être à ce jour que validées théoriquement. Il serait en effet impossible d'effectuer une évaluation psycho-ergonomique de DARE, étant donné par exemple, les travaux qui doivent encore être réalisés au niveau des interfaces homme-machine (IHM) pour les applets méta.

Cependant, il nous est possible d'étudier l'impact de DARE sur les processus de création, de mise en œuvre, et de maintien des supports d'activité qu'il fournit. En d'autres termes, quel est l'impact de nos travaux sur le processus de conception d'un collecticiel particulier ?

8.2.1 Le processus de conception classique

Nous avons déjà évoqué le processus classique de conception des collecticiels dans le chapitre 2. Ce processus est schématisé dans la Figure 56. On peut distinguer une certaine dichotomie entre les acteurs impliqués dans l'activité et ceux impliqués dans la méta-activité correspondante. Le sujet apparaît largement dans l'activité alors que le spécialiste en sciences humaines et l'informaticien sont plus présents dans la méta-activité. Ce processus peut être divisé en trois étapes distinctes : une collecte des besoins du sujet permettant l'analyse de l'activité, la récupération d'un ensemble de consignes servant à modéliser un support d'activité, et l'implantation du collecticiel dans l'activité du sujet, parfois accompagnée d'une phase d'évaluation en collaboration avec les spécialistes des sciences humaines. Lorsqu'un problème survient lors de l'utilisation du collecticiel, le processus effectue une nouvelle boucle. La critique du collecticiel précédemment implanté peut servir de base à la nouvelle collecte des besoins. Il se peut que le processus effectue plusieurs autres boucles avant l'acceptation du collecticiel par les sujets, ou avant son rejet définitif.

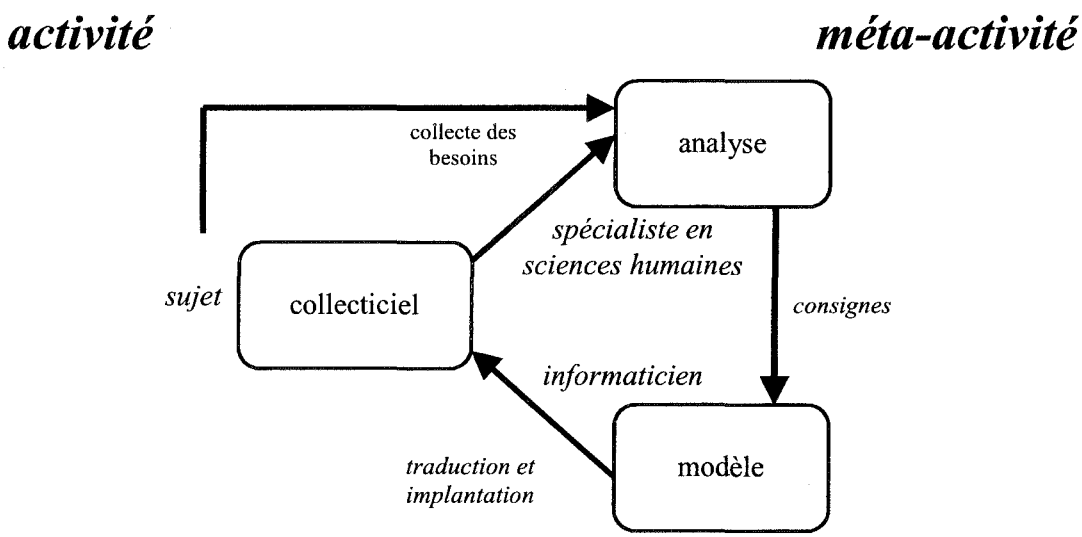

Figure 56. Le processus classique de conception d'un collecticiel

L'analyse et le modèle sont du domaine de la méta-activité. L'utilisation du support d'activité fourni par le collecticiel est du niveau de l'activité. Comme nous l'avons déjà largement présenté, nous pensons que cette dichotomie est néfaste à l'atteinte de l'objectif de ce processus, c'est-à-dire, à l'acceptation du collecticiel par les sujets. Les liens entre l'activité et son niveau méta sont faibles. Ce sont principalement les spécialistes en sciences humaines qui

font une incursion dans l'activité du sujet pour l'écouter et analyser la situation. Puis le modèle est généré (sans le sujet) et l'informaticien vient à son tour dans l'activité pour implanter son produit. Cette découpe implique généralement un grand nombre d'itérations de la boucle de conception, du fait des incompréhensions entre des acteurs qui ne partagent pas la même situation.

8.2.2 La conception centrée sur le sujet

Les liens entre les deux niveaux de l'activité peuvent être renforcés par une démarche du type conception participative [118]. Dans la conception participative, le sujet est plus grandement impliqué dans la méta-activité. Nous avons exposé cette approche au début de cette thèse (cf. chapitre 2). Elle est schématisée par la Figure 57 : le sujet devient central dans la boucle de conception. Il peut être complètement impliqué non seulement dans l'analyse mais aussi dans la modélisation du collecticiel. Les moyens utilisées sont classiquement des listes de contrôle, des scénarios et des prototypes [11].

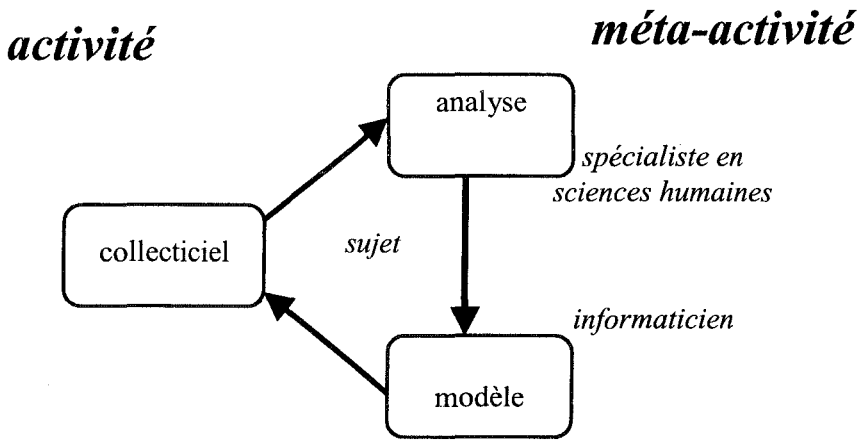

Figure 57. Le sujet au centre de la boucle de conception

La conception participative, en recentrant la boucle de conception sur le sujet, permet d'augmenter la compréhension entre les spécialistes (en sciences humaines et en informatique) et les sujets. Le processus de conception devient un processus plus coopératif.

Toutefois, cette démarche telle qu'elle est généralement mise en œuvre ne correspond pas réellement à effectuer une spécification des besoins in situ. La méta-activité est toujours faiblement connectée à son activité. Le lien est réalisé par le sujet qui passe d'une situation

d'utilisation du collecticiel à une situation de redéfinition de celui-ci. Il y a toujours deux situations différentes : une pour l'activité et une pour la méta-activité.

8.2.3 DARE : Le sujet et son activité au centre de la boucle

Le processus de conception est un processus coopératif. Le collecticiel est support au travail coopératif. DARE propose d'utiliser le support à la coopération offert par le collecticiel pour supporter sa propre (méta) activité coopérative de spécification. Ceci est rendu possible grâce à la satisfaction de nos enjeux vue précédemment.

Cette nouvelle donne introduit un nouveau schéma pour le processus de conception de collecticiels : ce sont à la fois le sujet et son collecticiel qui sont centraux. Ce nouveau processus est représenté Figure 58. La nouvelle approche adoptée peut laisser supposer qu'il n'y a plus besoin de spécialistes en sciences humaine ou en informatique dans le processus de conception. Cela est faux.

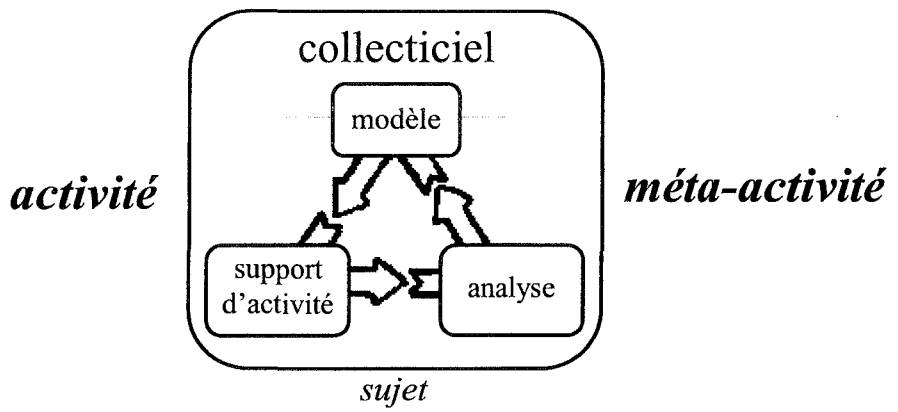

Figure 58. Le sujet et le collecticiel au centre de la boucle de conception

Dans ce nouveau processus de conception, les informaticiens, les spécialistes en sciences humaines, et les autres, sont devenu des sujets à part entière de l'activité, qui a été fusionnée avec sa méta-activité. Cette fusion donne une plus grande importance aux deux sous-systèmes principaux de DARE : Le système d'exploitation de supports d'activité et le dépôt de composants.

La partie que nous avons principalement détaillée dans cette thèse correspond au système d'exploitation des supports d'activité. En partant des mécanismes proposés par AT, il était

important de fournir les bases d'un système permettant l'*expansivité* du collecticiel (cf. chapitre 3). En fusionnant la méta-activité et son activité et en plaçant tous les protagonistes du processus de conception au même plan, DARE peut être vu comme un environnement de développement coopératif placé au sein de l'environnement d'utilisation de ses propres produits.

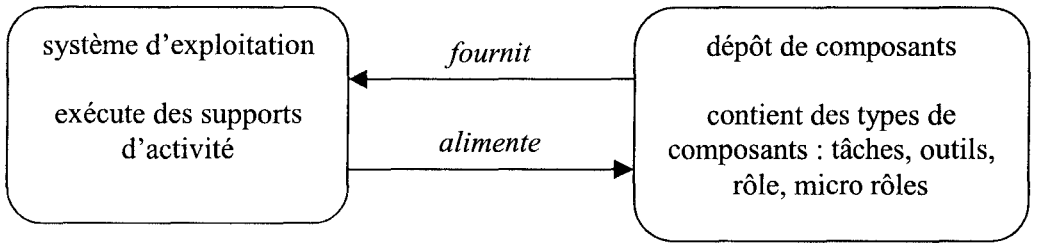

Figure 59. Les deux facettes de DARE

Cette nouvelle approche donne donc une grande importance au dépôt des composants. Le système d'exploitation de DARE utilise des composants du dépôt pour exécuter des supports d'activité. En retour il alimente le dépôt de composants (cf. Figure 59), puisque les développeurs, assembleurs et utilisateurs de composants sont tous impliqués dans les supports d'activité.

Actuellement, le dépôt de composants correspond à la machine virtuelle de Smalltalk. Smalltalk contient lui-même un dépôt de classes. Ce dépôt est l'objet Smalltalk qui est en fait le dictionnaire du système contenant toutes les classes existantes dans l'environnement de programmation. Certaines de ces classes sont les composants de DARE. Toutefois, maintenant que nous avons posé les bases nécessaires à la co-construction des composants au sein du collecticiel, il nous faut maintenant travailler, entre autre, aux moyens de stockage et de recherche de composants dans un dépôt qui peut être centralisé ou distribué, et alimenté par différentes activités. Nous verrons dans les parties 8.5.2 et 8.5.3 que de nouveaux travaux ont été amorcés à ce niveau.

8.3 Bilan des moyens mis en œuvre

Comme nous l'avons vu, DARE met en œuvre un nombre important de moyens qui proviennent de domaines parfois très différents. Pour ne citer qu'un exemple, sa réalisation touche aussi bien à l'étude des systèmes réflexifs, qu'aux sciences humaines, ou encore aux technologies de l'Internet. Cette partie tente de déterminer les bienfaits et méfaits des différents choix qui ont été arrêtés au cours de l'analyse, de la conception et de la réalisation de DARE.

8.3.1 La théorie de l'activité

L'AT nous a aidé à analyser le domaine de problèmes qui nous intéresse en nous permettant de définir les enjeux cités précédemment. Elle nous a de plus fourni les indices utiles à la réalisation du système. En particulier, le modèle d'Engeström a été crucial pour la mise en place de notre modèle conceptuel. Rappelons d'ailleurs que ce modèle est à la base du méta-modèle de DARE. Plus généralement, l'AT est le cadre conceptuel qui est à la base de l'architecture cadre à objets de DARE. Les travaux de Kuutti et Bardram nous ont orientés vers le fait que le collecticiel soit un outil à part entière. Cet outil fait partie de l'activité, et son utilisation comme sa redéfinition sont coopératives. Notons qu'à notre connaissance, DARE est le seul collecticiel à pousser aussi loin les concepts et mécanismes proposés par l'AT dans ses propres fondements.

Il faut signaler que l'AT se considère elle-même comme un artefact médiateur qui sert à penser. De ce fait, l'AT est encore aujourd'hui en construction. De nouveaux éléments sur les fondements de l'activité humaine devraient s'y greffer au fur et à mesure de son utilisation. C'est pourquoi, même si elle s'est révélée très utile, nous n'avons pas encore aujourd'hui découvert dans l'AT tous les fondements utiles à notre conception.

8.3.2 Les problèmes du modèle et du méta-modèle

Nous avons défini les collecticiels comme des supports à l'activité coopérative est non pas comme l'activité coopérative elle-même. La différence est importante. Ceci signifie que, quel que soit le système de TCAO mis en œuvre, une partie de l'activité reste chez l'humain. Nous pensons que *l'objectif d'un collecticiel n'est pas de modéliser l'activité*, ce qui semble

impossible du fait de sa complexité et des influences externes qu'elle subit, mais de lui *fournir un support adéquat.*

Ceci soulève une autre question. Dans un support d'activité coopérative, faut-il tout automatiser ? Notre culture d'informaticien nous entraîne à vouloir tout contrôler, demandant aux utilisateurs de s'adapter aux besoins du système. Les sujets doivent généralement spécifier un grand nombre d'informations nécessaires à ce contrôle. Un exemple flagrant est celui présenté dans le chapitre 1 : pour pouvoir utiliser ODESCA, il faut que l'utilisateur prévoie tous les cas de figures, planifie totalement son activité... DARE introduit une démarche opposée : le système s'adapte aux besoins des utilisateurs. Pour les composants du système, une spécification basique peut permettre de démarrer simplement une activité coopérative. Puis, ces spécifications peuvent être modifiées par les sujets qui veulent augmenter le contrôle du système sur leur support d'activité.

Dans cette optique, un méta-modèle de support d'activité simple peut suffire à spécifier les besoins des utilisateurs. Cependant, il ne faut pas non plus que ce méta-modèle ne permette pas d'augmenter le contrôle du système si le besoin s'en fait sentir. C'est pourquoi nous pensons que le méta-modèle de support d'activité actuel reste à être travaillé.

Dans nos travaux, le méta-modèle et le modèle de rôle que nous avons présentés sont encore assez faible. Par exemple, le modèle de rôles fonctionnels mis en jeu dans ODESCA (cf. chapitre 1) est bien plus complexe. Dans DARE, les actions ne peuvent aujourd'hui être exécutées qu'au niveau des micro-rôles. Or, un de nos objectifs est de prendre en compte les actions d'un sujet sur un outils particulier (au niveau d'un micro-rôle) pour en tirer des conséquences dans toute l'activité. Pour ce faire, il faut que les rôles ne soient plus simplement une collection de micro-rôles, mais qu'ils possèdent aussi un comportement leur permettant d'affecter tous les outils d'un sujets, ainsi que les rôles et outils des autres sujets dans l'activité. Dans cette première version de DARE, nous n'avons pas voulu mettre en œuvre un méta-modèle et un modèle de rôle trop complexe. Notre objectif principal, aux vues de AT, était de créer un système permettant de créer ou modifier des rôles à partir d'un modèle générique et grâce à l'introduction d'un méta-modèle. C'est ce que nous avons fait. Néanmoins, il nous faut aujourd'hui déterminer un modèle plus détaillé et y associer un moyen d'expression qui permette aux sujets de spécifier des rôles simplement. Ces points font partie des travaux que nous avons récemment entamés et sur lesquels nous reviendrons dans le paragraphe 8.5.1 de ce chapitre.

De la même manière, le méta-modèle et le modèle de la tâche demandent à être étendus. En effet, ceux-ci ne permettent pas de prendre en compte les contraintes pouvant exister dans une tâche. Par exemple, il est actuellement impossible dans DARE de spécifier précisément une tâche comme par exemple une partie d'échecs. Dans une telle tâche, il y a un outil qui correspond à l'échiquier, le rôle du joueur et le rôle de l'arbitre. Ces éléments peuvent être spécifiés dans notre environnement. Par contre, il est impossible que DARE prenne en compte le fait qu'il n'y ait qu'un seul arbitre et exactement deux joueurs dans une partie d'échecs. Un autre problème que nous avons déjà évoqué dans le chapitre 5 concerne les contraintes pouvant exister entre les sous-tâches d'une tâche. Ces problèmes rejoignent ceux du Workflow et ne sont pas triviaux. Il faut que le système puisse déterminer quand un support d'activité lié à une tâche peut démarrer, si plusieurs tâches peuvent être réalisées en parallèle, etc. Il faut aussi réfléchir aux moyens d'expression à proposer pour spécifier de telles contraintes. Au niveau de l'Interface Homme-Machine, celles-ci peuvent être spécifiées graphiquement ou dans un langage textuel particulier. Il reste donc de nombreuses recherches et de nombreux travaux à intégrer dans DARE. Néanmoins, nous pensons que les bases que nous avons proposées faciliteront la mise en place de tels mécanismes. Il s'agit d'une part d'étendre le modèle et le méta-modèle de DARE tout en reposant sur les mécanismes déjà en place. D'autre part, pour les Interfaces Homme-Machine, il sera nécessaire de définir de nouvelles applets méta à intégrer dans l'environnement.

8.3.3 La malléabilité

L'étude de l'AT et les enjeux que nous nous sommes fixés nous ont guidés vers le domaine de recherche de la malléabilité. Ce domaine s'intéresse principalement aux moyens permettant la modification des logiciels par leurs propres utilisateurs. Notre intérêt dans ce domaine était nécessaire car, selon les enjeux fixés, DARE apparaissait comme un logiciel malléable. Cependant, avec le recul, il nous semble encore plus logique que la malléabilité ait été une des facettes de l'étude des problèmes liés à DARE.

La malléabilité est un domaine hybride à la jonction entre le monde de l'utilisateur, donc de l'activité humaine, et le monde de l'informatique. Le paradoxe de la malléabilité est de vouloir permettre le développement d'un outil informatique par ses propres utilisateurs : ceux-ci ne sont généralement pas des développeurs. C'est pourquoi la malléabilité propose des solutions antagonistes qui peuvent difficilement faire l'unanimité. D'un côté, l'outil informatique est aisément modifiable par l'utilisateur, mais l'étendue des modifications

possibles est limitée. D'un autre côté, l'outil informatique est totalement redéfinissable, mais les modifications demandent à l'utilisateur une grande expertise. DARE n'a pas échappé à ce dilemme. Pour tenter de résoudre ce problème, la solution que nous avons mise en place est (1) de limiter les possibilités de modification du système au domaine problème qui nous intéresse, et (2) de fournir aux utilisateur les moyens de trouver différents points d'équilibre en fonction de leurs motivations et compétences.

Le point (1) a été réalisé par l'introduction d'un nouveau méta-modèle basé sur notre modèle conceptuel. Ainsi, nous avons tenté, autant que possible, d'élever les niveaux d'abstraction du système en passant d'un méta-modèle objet[1] à un méta-modèle basé sur les concepts et mécanismes de AT. Pour le point (2), nous avons introduit dans le méta-modèle de DARE différents mécanismes de modifications des modèles. En particulier, les instances des méta-types peuvent être manipulés soit par intégration, soit par extension. Il existe aussi une certaine malléabilité hiérarchique permettant d'accéder à différents niveaux d'abstraction.

Il est à noter qu'un moyen puissant permettant d'élever un peu plus le niveau d'abstraction général de notre méta-modèle n'as pas encore été réellement exploité dans DARE : il s'agit de l'utilisation de métaphores au sein des Interfaces Homme-Machine (IHM). Nous reviendrons sur ce point en 8.3.9. Néanmoins, les points (1) et (2) présentés ci dessus fournissent les bases nécessaires à tout nouveau développement. Ces points ont été implémentés par une approche composant et la création de l'architecture cadre de DARE.

8.3.4 Les composants

Nous sommes convaincus que les composants sont l'avenir de l'informatique en général et du TCAO en particulier. D'ailleurs nous ne sommes pas les seuls si l'on considère le nombre de publications actuelles qui en font état, ainsi que les efforts faits en ce sens par des consortiums comme l'OMG. Les composants sont souvent plébiscités du fait de leur capacité à favoriser la réutilisation. Du point de vue informatique, il s'agit de la réutilisation de code fiable. Nos travaux dans la réalisation de DARE se trouvant au point de convergence de l'informatique et de l'étude de l'activité humaine, nous apportons un autre point de vue sur les propriétés des composants.

Qu'est-ce que la réutilisation de code fiable ? C'est réutiliser du code qui a été testé, modifié et amélioré pour un certain usage. De notre point de vue, il s'agit là de réutiliser pour un

problème similaire l'expérience d'autres utilisateurs de ce code. En fait, le composant est une encapsulation de l'héritage historique et culturel de ceux qui ont contribué à son développement. De plus, le composant, de par l'encapsulation, permet de tirer profit de cette expérience sans avoir obligatoirement à se soucier de ce qu'il contient.

Cependant, un composant n'est pas simplement fiable et réutilisable. Il a été construit et reconnu utile dans une certaine activité. En d'autre termes, il cristallise l'héritage historique et culturel d'une certaine situation. Par conséquent, il est généralement inutile, peu adapté ou incompréhensible dans une situation trop différente. En informatique, ce dernier point a une forte résonance sur le modèle du composant.

Le modèle d'un composant définit, entre autre, son interface d'utilisation. Le modèle est créé pour un certain type d'activité et, même si le contenu du composant peut s'avérer utile dans un contexte différent, il est difficile de l'utiliser dans une activité différente. Nous avons nous même souligné et été confrontés à ce type de problèmes dans la réalisation de DARE. En effet, DARE a vocation à intégrer des applets Java pour les lier aux supports d'activité des sujets. Nous avons exposé comment il est techniquement possible de réutiliser et lier finement des (composants) applets à DARE. Cependant, cette tâche est rendue très difficile car les applets sont des composants destinés à être intégrés dans un environnement d'exécution tel un navigateur Internet, et pas spécialement dans un environnement de TCAO. Le contexte d'utilisation pour lequel le modèle des applets a été créées est différent du notre, ce qui les rend difficilement réutilisable pour nos besoins. Ceci explique que nous ayons commencé de nouveaux travaux concernant la définition d'un modèle de composant du type applets pour le TCAO. Nous reviendrons sur ce point en 8.5.2.

Toutefois, DARE met en œuvre d'autres composants que les applets. Ce sont les instances des méta-types de son méta-modèle, c'est-à-dire les tâches, outils, rôles et micro-rôles. L'utilisation de ces composants dans DARE est simplifiée car leur modèle a été conçu dans le contexte de DARE et pour fonctionner dans un cadre bien défini. Ce cadre est implémenté sous la forme de l'architecture cadre de DARE.

8.3.5 L'architecture cadre

Nous ne pensons pas qu'il aurait été possible de construire DARE sans la réalisation d'une architecture cadre. Par définition, une architecture cadre définit un cadre qui spécifie les liens

[1] Défini en termes de classe, d'association, etc.

statiques et les liens modifiables entre les éléments des applications qu'elle supporte. Ces différents liens correspondent, dans le jargon lié aux architectures cadres, aux point froids et points chauds. Le principe de base de DARE est de fournir des supports d'activité modifiables par les sujets. Ceci implique la définition précise des points froids et points chauds d'un modèle de support d'activité, ainsi que les moyens de modification de ces derniers.

Une architecture cadre implémente un modèle générique qui, pour être mis en œuvre, impose un surcroît de travail par rapport au développement d'une application classique. Cette majoration n'est fondée que lorsqu'il est prévu d'implémenter plusieurs spécialisations du modèle générique [106]. Dans DARE, le modèle générique implémenté est celui du support d'activité. De ce fait, nous identifions deux raisons majeures qui fondent notre approche. Premièrement, chaque modèle de support d'activité, correspondant à une tâche spécifique, est appelé à être modifié par ses utilisateurs : le modèle générique est le lien entre un modèle spécialisé et un autre. Secondement, le modèle générique de support d'activité est un modèle récursif : toute tâche peut contenir d'autres tâches. De ce fait, DARE met en même temps en œuvre plusieurs spécialisations, souvent différentes, du modèle générique.

Un autre débat concernant l'architecture cadre concerne la mise en place d'une architecture cadre boîte blanche, d'une architecture cadre boîte noire, ou d'un peu des deux : une architecture cadre boîte grise. Nous avons vu que DARE est une architecture cadre boîte grise. Nos réflexions post-réalisation à ce sujet sont proches des réflexions liées au choix de l'introspection et/ou de l'intercession dans les mécanismes réflexifs mis en œuvre. C'est pourquoi nous reviendrons sur ce problème en 8.3.7.

D'une manière générale, les architectures cadres informatiques s'adressent aux informaticiens. Ceci est aussi vrai lorsque les composants sont des éléments graphiques pouvant être assemblés visuellement. Du fait de nos enjeux, DARE se différencie d'une mise en œuvre classique en laissant les sujets impliqués dans un support d'activité accéder au framework sous-jacent. Pour ce faire nous avons choisi de rester proche des fondements de l'AT et avons lié l'approche framework à celle de l'implémentation ouverte.

8.3.6 L'implémentation ouverte & le MOP

Les composants d'une architecture cadre, ou encore le modèle générique sous-jacent, peuvent être d'un haut niveau d'abstraction. Ceci signifie que les composants et/ou modèle générique des applications supportés sont proches du domaine d'utilisation visé. C'est le cas pour

DARE : les composants tels que les tâches, outils et rôles sont du domaine du support à l'activité coopérative. Le modèle générique de support d'activité l'est lui aussi. Cependant, le haut niveau d'abstraction de ces éléments ne signifie pas forcément que les moyens de redéfinition de l'application le sont aussi.

L'approche adoptée par l'implémentation ouverte offre une solution élégante pour définir les moyens de la malléabilité. L'interface de base permet l'utilisation de l'application, l'interface méta permet sa redéfinition en introduisant un protocole bien établi. Nous pensons que notre approche orientée framework devient originale de par sa liaison à celle de l'implémentation ouverte. Cette démarche nous a permis de définir un moyen d'accès à la malléabilité, c'est-à-dire à la redéfinition et /ou recomposition des éléments de l'architecture cadre dans un protocole inspiré de AT. Ainsi, pour redéfinir une tâche, le sujet ne modifie pas la classe sous-jacente, mais y ajoute des outils ou des rôles. Bien entendu, à un niveau plus bas d'abstraction, il s'agit bien d'une modification de classe. Mais nous pensons que l'interface méta mise en place permet de fournir des bases solides à de futurs développement, aussi bien de la part des sujets, que de développeurs plus spécialisés (pour les IHM par exemple). Rappelons que notre principal objectif, dans le cadre de cette thèse, était de construire ces bases.

Une autre remarque sur l'implémentation ouverte nous semble intéressante. Dans sa propre définition, l'implémentation ouverte est très proche des problèmes de notre domaine. En effet, en séparant l'interface de base de son interface méta, elle expose une indéniable prise en compte des deux dimensions de l'activité : le niveau de sa réalisation et son niveau méta (la méta-activité). De plus, elle permet de définir les moyens d'accès à l'expérience cristallisée dans les artefacts de l'activité. L'interface méta, suivant sa définition permet ainsi aux sujets qui utilisent un composant de s'approprier l'expérience qu'il contient, et éventuellement, d'y apporter la sienne.

Aujourd'hui, l'implémentation ouverte est fortement utilisée, en particulier dans les langages de programmation par le biais du MOP. Ces conclusions nous amènent à identifier *une indéniable percée des fondements de l'activité humaine dans les fondements de l'informatique*. Cette percée s'est peut être réalisée d'une manière intuitive, mais semble être une des clefs de l'avenir de l'informatique en général. Il existe d'ailleurs aujourd'hui dans l'informatique d'autres mouvements, comme l'introduction de la modélisation par les rôles, qui suivent plus ou moins consciemment cette voie. La démarche que nous avons adoptée pour nos travaux suit ce mouvement de manière explicite et recherchée : *nous tentons*

d'utiliser les fondements de l'activité humaine pour définir de nouveaux artefacts informatiques.

8.3.7 Un système réflexif

L'implémentation ouverte nous a amené à construire un système réflexif lui-même basé sur des langages réflexifs : Java et Smalltalk. Dans un système comme le nôtre, la réflexivité est un passage obligé. Pour pouvoir modifier un système, qui plus est, au cours de sa propre exécution, il faut fournir un accès à son implémentation, c'est-à-dire à son modèle d'exécution. L'utilisation de DARE étant intimement liée à l'accès à son implémentation, il doit être un système réflexif. Plus généralement, les systèmes basés sur une approche composants sont des systèmes réflexifs. On peut citer en exemple, de par leur notoriété, les JavaBeans. Ce type de composant repose principalement sur les mécanismes réflexifs de Java.

Toutefois, nous avons déjà souligné que la réflexivité possède deux facettes. L'une est l'introspection, l'autre est l'intercession. Java ne permet que l'introspection, Smalltalk permet à la fois l'introspection et l'intercession. D'un manière générale, l'introspection permet d'examiner un système, alors que l'intercession permet de le modifier. Dans DARE, nous avons utilisé les deux mécanismes.

Le problème de l'intercession est d'exiger une mise en œuvre lourde. L'intercession est l'apanage des langages interprétés qui, s'ils sont très souples, ont la particularité d'être complexes et généralement plus lents à l'exécution que les langages compilés. De plus, les systèmes basés sur des langages interprétés sont réputés comme difficiles à maintenir [21] : lorsque les utilisateurs peuvent redéfinir le système à leur guise. La question qui se pose alors est pourquoi l'intercession ?

Selon notre expérience dans la mise en œuvre de la réflexivité dans DARE, nous concluons que l'introduction de l'intercession dépend du niveau d'abstraction des composants disponibles et du niveau d'abstraction des modifications désirées.

Dans le cadre d'une architecture cadre, l'introspection et l'intercession prennent une dimension qui peut sembler quelque peu différente des définitions données ci-dessus. L'introspection est suffisante pour étendre l'application qui repose sur ce framework. Elle permet d'examiner les composants et de les lier au reste du système. C'est ce qui est fait dans les JavaBeans. Elle peut même permettre de modifier les composants eux-mêmes, en

supposant que ceux-ci soient construits hiérarchiquement, c'est-à-dire à partir d'autres composants. Ce sont les architectures cadres boîte noire.

L'intercession quant à elle peut agir à plusieurs niveaux. D'un côté elle peut être utilisée pour redéfinir l'architecture cadre. En d'autres termes, il s'agit de modifier l'ensemble des points froids et/ou des points chauds, et/ou des mécanismes qui y sont associés. On modifie alors les fondements de l'architecture cadre, le modèle générique sous-jacent. Toutefois, ceci ne nous intéresse pas dans DARE. L'intercession est obligatoire lorsqu'on veut créer ou modifier des composants qui ne sont pas eux-mêmes constitués de composants. On parle alors de framework boîte blanche.

Apparaissent alors les deux dimensions qui impliquent l'utilisation de l'un ou de l'autre de ces mécanismes. Lorsqu'on veut pouvoir créer ou modifier des composants en manipulant d'autres composants, l'introspection suffit. Pour créer ou modifier des composants directement liés au langage d'implémentation, l'intercession est nécessaire. Par exemple, pour ajouter une méthode à la classe d'un objet, on utilise généralement l'intercession. Toutefois, si la méthode existe déjà sous forme de composant et peut être intégrée à l'objet, il suffit alors de l'introspection. Pour résumer, si les modifications du système désirées à un certain niveau d'abstraction peuvent être réalisées à partir de l'intégration (ajout, retrait et liens) de composants correspondant à ce niveau, l'introspection suffit.

La plupart des systèmes malléables qui apparaissent aujourd'hui dans notre domaine de recherche n'utilisent que l'introspection [55][54][124]. Ces réalisations peuvent donc être implémentées en Java, en utilisant un chargement dynamique de classes dans l'environnement d'exécution de l'application modifiée. Ces systèmes ne fonctionnent que par intégration. Toutefois, leurs auteurs concluent généralement que leur framework est opérationnel mais que les composants pouvant y être intégrés doivent encore être développés.

Dans DARE, certains composants doivent eux aussi préexister aux supports d'activité : ce sont les applets. A l'inverse, les autres composants du systèmes peuvent préexister, ou être créés pour l'occasion. Nous avons tenté de rester au plus proche des fondements proposés par l'AT. Notre but est de fournir aux premiers utilisateurs de DARE des composants basiques que nous aurons nous même créés et de laisser ces sujets les affiner au cours de leurs activités. Dans les premiers temps, ces modifications ne pourront pas être faites par recomposition d'autres composants, car ceux-ci ne seront pas assez nombreux et diverses pour répondre aux besoins. C'est pourquoi l'extension et l'intercession sont à notre avis nécessaire, au début. A

l'inverse, nous espérons qu'au fur et à mesure de l'utilisation de DARE, le nombre de composants développés et reconnus utiles pour certains types d'activités par leurs propres utilisateurs sera assez conséquent pour former une sorte de noyau minimal. Celui-ci permettra alors la satisfaction des nouveaux besoins simplement par intégration.

Sur ce point nous rejoignons tout à fait les idées de Reenskaug, père de l'architecture cadre MVC (Modèle Vue Contrôleur) et de la méthode de l'ingénierie logicielle OOram[1] [109]. Reenskaug déclare [110] que toute être vivant est constitué de protéines. Il existe des millions de protéines différentes. Cependant, toute protéine n'est qu'une composition réalisée à partir de trente deux acides aminés. Le rêve de Reenskaug est de pouvoir créer autant de systèmes informatiques qu'il existe d'êtres vivants à partir de simplement trente deux architectures cadres[1] réutilisables. Notre rêve est de pourvoir supporter tout type d'activité coopérative à partir d'un nombre minimal de composants, développés par les sujets et réutilisés par intégration. A ce moment, les mécanismes lourds et difficiles à utiliser de l'intercession, liés à l'extension, pourront certainement être abandonnés.

8.3.8 Architecture distribuée sur Internet

L'Internet (accès par un navigateur banalisé) nous semble toujours être la solution la plus adéquate pour fournir un accès uniforme au collecticiel, ainsi que pour supporter la mobilité des utilisateurs. Nous avons pu malgré tout déceler plusieurs problèmes liés à cette démarche.

Pour que la solution Internet soit viable, il faut que les applets soient de petites applications. Lorsque celles-ci sont trop lourdes, les temps de chargements peuvent s'allonger et lasser les utilisateurs. Les principales applets que nous avons réalisées et utilisées pour tester les fonctionnalités du noyau de DARE sont les applets méta. Comme nous l'avons vu, ces applets utilisent CORBA pour communiquer avec le serveur d'activités. Le navigateur que nous avons utilisé est Netscape Communicator 4.07 qui contient l'ORB Visibroker pour Java. Malheureusement, l'ORB de Netscape ne permet d'instancier que des clients CORBA. Or nos applets fonctionnent à la fois comme client (pour récupérer les informations sur le support d'activité) et comme serveur (pour la mise à jour en temps réel des informations). Il nous a donc fallu utiliser un ORB plus complet. Cet ORB est une autre version de Visibroker pour Java. Cette nouvelle version est téléchargée dans le navigateur des sujets avec chaque applet pour que celle-ci soit opérationnelle : le fichier archive correspondant a une taille de 2248 Ko.

[1] Nous introduisons cette méthode dans le paragraphe 8.5.1.2

De plus, nos applets méta utilisent Swing, la nouvelle API graphique intégrée dans Java 2. Le navigateur que nous utilisons utilise la version 1.1.8 de Java. Les classes Swing n'y sont pas présentes. Il faut alors les télécharger : le fichier archive correspondant a une taille de 2254 Ko. Chaque applet méta est donc accompagné, au minimum d'une archive de 4502 Ko ! A cela vient s'ajouter la relative lenteur de Java pour démarrer l'applet dans le navigateur. En moyenne, le temps nécessaire pour démarrer la première applet méta sur un processeur Pentium 75 MHz dans un réseau 10 Mb était en moyenne de 5 minutes. Il est possible d'alléger la charge de données qui doivent être téléchargées. Un premier moyen est d'installer les archives directement sur le poste client dans le chemin des classes Java du navigateur. Un autre moyen est d'installer des plug-ins. Toutefois, ceci revient à modifier le poste client et donc à perdre le bénéfice de la mobilité de l'utilisateur.

Un autre point concerne la gestion de la sécurité au niveau des applets. Celle-ci pose des problèmes pour construire un environnement de travail conséquent. Il est par exemple difficile de trouver un bloc note sous forme d'applet, étant donné que, si elles ne sont pas signées, celles-ci ne peuvent accéder au disque local du poste du sujet. Ce point est une des limites pouvant être rencontrées par les sujets qui veulent créer un support d'activité 'idéal'. Un autre point concerne les communications réseau. En effet, une applet ne peut établir des connections réseau qu'avec la machine hébergeant le serveur HTTP qui a servi le document qui la référence. Dans le cadre de DARE, ceci implique que le serveur d'activités soit sur la même machine que le serveur HTTP fournissant les applets méta. Ce point peut aussi poser des problèmes pour les applets coopératives ou partagées liées à DARE.

Enfin, nous avons pu constater avec dépit que les navigateurs Internet ne sont standards que jusqu'à une certaine limite. Il nous a par exemple été impossible de faire fonctionner DARE à partir d'Internet Explorer (MIE) de Microsoft. En particulier, la mise en œuvre de CORBA dans ce navigateur est totalement différente de celle de Netscape. Il nous a en effet fallu utiliser des tags spéciaux pour forcer Netscape Navigator à utiliser un ORB différent de celui par défaut. Ces tags ne sont pas valides dans MIE.

Nous n'avons à ce jour pas de solution idéale à ces problèmes qui relèvent plus de l'ingénierie que de la recherche.

[1] Dans le sens d'architectures cadres OOram

8.3.9 Le problème des IHM

Ce bilan expose les expériences que nous avons pu tirer de notre réalisation. Toutefois, il manque à ce bilan une réponse importante : ces moyens sont-ils réellement adaptés à l'utilisateur. Seule une évaluation psycho-ergonomique de DARE permettrait d'y répondre. Nous n'avons malheureusement pas pu procéder à cette évaluation pour une raison simple : nos travaux ne constituent que les bases à la création d'un environnement complexe touchant à des domaines très différents. Nous nous sommes ici principalement intéressés à la création de l'architecture de DARE en tentant de l'ancrer au maximum dans les fondements des sciences humaines et informatiques que nous pensons adéquats. En particulier, les applets présentées (les applets méta) n'ont été développées que pour tester l'architecture sous-jacente du système. Même si notre framework, le modèle générique d'activité, et le méta-modèle de DARE élèvent le niveau d'abstraction du système vers les sujets, un gros travail reste à faire du côté des Interfaces Homme-Machine (IHM).

Une IHM adaptée peut masquer une certaine complexité en effectuant une projection visuelle des mécanismes du méta-modèle sous-jacent. Il est par exemple plus aisé de lier des composants graphiquement (comme dans la BeanBox de Java) qu'en appelant les opérations de l'interface IDL du méta-modèle. Une IHM influence fortement l'activité du sujet face à son outil. Les IHM sont un domaine de recherche à elles seules. Nous n'en sommes pas spécialistes. Elles requièrent d'autres capacités que celle de la création de systèmes réflexifs et d'architectures ouvertes et distribuées. Nous envisageons des collaborations avec de tels spécialistes pour rendre DARE plus facilement utilisable qu'il ne l'est à présent. Alors seulement, nous serons en mesure d'évaluer précisément la qualité du support à l'activité coopérative fourni par DARE à partir des fondations que nous venons de poser. Ces collaborations devraient s'avérer d'une mise en œuvre simple, étant donné que notre laboratoire d'accueil (Trigone) possède un tel axe de recherche ainsi qu'une équipe de chercheurs spécialisés en ergonomie et dans les IHM.

8.4 Comparaison de nos résultats aux travaux analogues

Les chapitre 2 de cette thèse a présenté les travaux se rapprochant le plus de ceux que nous avons mené dans le cadre de DARE. Nous aimerions toutefois revenir sur la spécificité de notre contribution par rapport aux thèmes abordés. Ces principaux thèmes sont les suivants : les fondements des systèmes de TCAO, la malléabilité, et les architectures distribuées.

8.4.1 Architecture distribuée

Nous ne pensons pas avoir contribué avec DARE au développement de nouvelles technologies servant le domaine des architectures distribuées. Notre incartade dans ce domaine s'est limité à utiliser des technologies de plus en plus matures telles que CORBA. Nous n'avons d'ailleurs pas utilisé ces technologies au maximum de leur potentiel. Chaque support d'activité est lui-même distribué entre les postes des différents sujets qui y participent. Même s'il ne l'est actuellement pas, le serveur d'activités pourrait lui aussi être distribué. Le logiciel utilisé (Distributed Smalltalk) est d'ailleurs tout à fait capable de supporter une telle distribution. Nous n'en avons pour l'instant pas encore ressenti le besoin ni l'utilité. Toutefois, cette question reste à creuser. De plus, notre utilisation de CORBA peu se révéler assez faible comparée aux nombreux services que ce standard définit et que nous n'avons pas désiré utiliser.

Néanmoins, DARE utilise les mécanismes de base fournis par CORBA dans un contexte qui nous semble original : la programmation à distance d'un système réflexif et le pilotage d'applets Java liées dynamiquement au reste du système. Il s'agit de ce fait de la création d'un environnement distribué à la topologie sans cesse reconfigurée et en constante extension.

8.4.2 La malléabilité

Les systèmes malléables généralement proposés dans le TCAO correspondent à ce que dans DARE nous avons appelé des outils. Autrement dit, ils sont du niveau des collecticiels isolés définis dans le Chapitre 1. La malléabilité est principalement considérée au niveau des composants au sein de cet outil, comme des composants graphiques. Chaque outil est malléable dans le cadre d'une micro-activité spécifique et bien définie. DARE permet d'intégrer plusieurs outils dans des supports d'activité et de définir des règles de coopération à ce niveau global. De ce fait, DARE agit à un niveau plus général de l'activité. S'il sont réalisés en Java sous forme d'applets, ces systèmes malléables peuvent même être intégrés dans DARE. Ainsi, dans un support d'activité, on aura accès à la malléabilité au niveau de l'activité globale gérée par DARE, et au niveau des applets liées si elles le proposent. S'il existe une API pour la malléabilité de ces applets, DARE sera même capable de déclencher les méthodes adéquates sur l'applet liée, et donc de gérer la malléabilité de cet outil : la modification de l'outil devient une action du micro-rôle associé.

Une spécificité de notre approche est d'identifier que le composant cristallise et permet la réutilisation d'un héritage historique et culturel de la situation. Notons toutefois que d'autres chercheurs du domaine appréhendent le composant dans une approche similaire, même si elle n'est pas présentée de cette manière. Par exemple, Morch définit un modèle de composants intégrant ce qu'il appelle de rationnel de conception [89][91]. En fait, chaque composant contient une description des altérations qu'il a subies pour arriver dans son état actuel. Dans DARE, le composant doit être construit par les sujets qui l'utilisent et peut être réutilisé par d'autres dans l'état. Sur ce point, ce qui nous démarque des approches classiques est que nous voulons non seulement que la communauté puisse adapter son support d'activité (par intégration ou même par extension), mais aussi que cette adaptation soit réutilisable et adaptable à nouveau par une autre communauté. De cette manière, l'expérience développée dans un certain contexte peut être bénéfique à un autre. C'est aussi pourquoi notre objectif n'est pas de peupler complètement nous-mêmes un dépôt de composants, mais de fournir les bases aux sujets pour qu'il le fassent eux-mêmes (par exemple, par la collaboration de spécialistes d'un domaine et d'utilisateurs de ce domaine impliqués dans un même support d'activité).

Un autre point qui nous démarque des approches classiques est d'offrir une base à la malléabilité par l'introduction d'une architecture cadre défini à partir des concepts et mécanismes tirés de AT. De ce fait, notre framework est moins basé sur l'apport d'une solution technique à la création d'un système malléable, que sur les fondements de l'activité humaine que DARE tente de supporter. L'utilisation de l'architecture cadre s'en trouve aussi affectée puisque l'accès à la malléabilité est réalisé de manière uniforme et dans un processus coopératif. Autrement dit, *les outils permettant la malléabilité sont des outils à part entière qui ne sont pas en marge du système.*

Enfin, nous avons mis en œuvre les possibilités d'intégration de n'importe quelle applet Java. Classiquement, les systèmes malléables utilisent des composants spécifiquement définis pour celui-ci. Notre démarche est à la fois une force et une limite. La force de DARE à ce niveau est de s'ouvrir à un type de composants standardisés pouvant être découvert par les sujets sur le plus grand système informatique ayant jamais existé : Internet. La limite de cette approche est que ce modèle de composant est trop général pour répondre aux besoins spécifiques du TCAO, et, de ce fait, ces composants sont généralement difficilement intégrables en l'état.

8.4.3 Les fondements des systèmes de TCAO

Notre démarche, par rapport au domaine de recherche du TCAO en général, s'inscrit dans une mouvance récente qui tente de fusionner les fondements de l'informatique et ceux des sciences humaines. L'enjeu est de réaliser des systèmes que nous espérons plus proches des besoins des utilisateurs, étant issus de cette fusion. Nous avons présenté dans le chapitre 2 quelques-unes des contributions d'autres chercheurs du domaine dont les travaux sont les plus proches des nôtres. Notre but est ici d'identifier quels sont les apports de nos travaux dans cette nouvelle voie.

Premièrement, les recherches sur les fondements des systèmes de TCAO semblent plus que jamais être d'actualité. Nous avons cité au début de cette thèse quelques-uns des différents récents appels à communication sur le sujet « Evolution des systèmes pour le TCAO ». Pourtant, même si l'AT est souvent considérée comme très prometteuse, il n'existe pas à notre connaissance de système de TCAO l'utilisant comme réel fondement. DARE constitue de ce fait la première réalisation d'un tel système. Nous avons montré la faisabilité d'une telle approche et la fusion opérée entre les fondements des sciences humaines et des techniques de conception logicielle avancées.

DARE n'est pas le seul système réflexif ayant été réalisé dans le domaine du TCAO. Toutefois, DARE se démarque des autres réalisations telles wOrlds [130] ou Prospero [31][36]. En s'appuyant sur AT, DARE propose ses mécanismes réflexifs à ses utilisateurs finaux. Ainsi, la réflexivité est réellement utilisable in situ, lorsque le besoin s'en fait sentir et dans un processus coopératif.

Cette réflexivité peut néanmoins introduire une certaine difficulté d'utilisation du système. C'est pourquoi nous avons en conséquence proposé un nouveau processus de conception pour le TCAO. Ce processus a pour particularité de placer tous ses acteurs au même plan. La conception étant une activité coopérative, à la fois le sujet (spécialiste ou utilisateur basique) et son collecticiel sont placés au centre de la boucle de conception. Le collecticiel est utilisé pour supporter l'activité coopérative de sa propre (re)définition.

Nos travaux fournissent les bases à l'achèvement d'un tel système. Ces bases sont solidement ancrées à la fois dans les fondements de l'activité humaine qui en sont à l'origine (AT), et dans les avancées du domaine de l'informatique en matière de techniques de conception logicielle (Architectures cadres, implémentation ouverte, MOP et systèmes réflexifs). Elles

sont d'ailleurs déjà utilisées dans de nouvelles contributions qui suivent cette nouvelle voie et élargissent la brèche que nous venons d'entrouvrir.

8.5 La poursuite des travaux

Cette partie présente les travaux que nous avons entamés dans la poursuite de DARE. Certains de ces travaux sont réalisés en collaboration avec d'autres chercheurs et jeunes chercheurs en informatique qui s'intéressent, de près ou de loin, au TCAO.

8.5.1 Les scénarios et les rôles

Nous avons soulevé le problème de la définition d'un modèle de rôles et des moyens permettant à l'utilisateur final de les spécifier. Ce problème rejoint le problème général qu'est celui de la spécification de la tâche. Spécifier une tâche revient à créer un modèle d'activité. Nous suivons actuellement deux pistes pour répondre à ce problème.

8.5.1.1 Une approche par les théories du langage

La première piste a été amorcée dans le cadre d'une nouvelle collaboration avec l'équipe Systèmes Communicants de l'Université de Savoie. Cette équipe travaille sur les moyens d'ajouter de la régulation dans les collecticiels [83]. Cette idée est très proche de la nôtre, en terme d'introduction du support à la méta-activité dans le collecticiel. Toutefois, à l'origine, ces travaux ne reposent pas sur les bases proposées par AT. Leurs fondements se trouvent dans les théories du langage. Leur proposition consiste en une approche par la création de scénarios d'utilisation servant de base à ce que nous appelons la spécification de la tâche. De ce fait, ils ont principalement travaillé sur les moyens d'expression à mettre en place dans les Interfaces Homme-Machine et qui font aujourd'hui défaut à notre réalisation. A l'inverse, ils ne possèdent pas de plate-forme telle que DARE, possédant les fondements adéquats à l'introduction de cette régulation dans un collecticiel. Cette collaboration devrait aujourd'hui permettre à chacun de tirer parti des avancées complémentaires de l'autre, et de proposer de nouveaux résultats de recherche.

8.5.1.2 Une approche par la méthode OOram

La seconde piste que nous suivons s'intéresse directement au domaine de la modélisation orientée objet en ingénierie logicielle. Il s'agit de la modélisation par les rôles. Notre approche se base principalement sur la méthode OOram [109] dont nous avons déjà brièvement parlé.

La modélisation par les rôles peut être considérée comme une réaction aux problèmes posés par les techniques de modélisation traditionnelles. Ces problèmes sont principalement liés au concept de classe. En particulier, les diagrammes de classes comme on peut les trouver dans UML permettent de spécifier les relations qui existent entre leurs instances dans le système à objets. Ces relations correspondent à une description des liens statiques entre les objets. A l'inverse, les diagrammes de collaboration permettent d'exprimer la dynamique existante entre les objets. Malheureusement, ces diagrammes sont réalisés en utilisant directement les objets instances des classes. Ainsi, la collaboration décrite entre deux objets est peu réutilisable. Si plusieurs objets instances de classes différentes mettent en œuvre un même schéma de collaboration, il est nécessaire de décrire plusieurs fois ce schéma.

La modélisation par les rôles propose d'abstraire les schémas de collaboration en introduisant la notion de rôle. Lorsqu'un objet participe à un schéma de collaboration particulier, on associe à sa classe le rôle correspondant dans le schéma identifié. Ainsi, il est possible d'identifier plusieurs schémas réutilisables de collaboration entre rôles. Le but du rôle est d'abstraire l'identité d'un objet dans un schéma de collaboration particulier. Un objet peut participer à plusieurs schémas de collaboration. La classe reste un concept puissant permettant la réutilisation de code. Le rôle est un concept puissant permettant la réutilisation de schéma de collaboration. Il supporte des mécanismes tels l'héritage-spécialisation, le polymorphisme, la composition, et l'agrégation. Un exemple de schéma de collaboration bien connu est celui de MVC (Modèle Vue Contrôleur). Dans un système objet, plusieurs objets de classe différentes peuvent jouer le rôle du modèle. Un objet particulier peut être à la fois être le modèle dans une certaine collaboration, et le client dans une autre qui implémente un schéma de type client-serveur.

La modélisation par les rôles s'intéresse donc à la modélisation de l'activité coopérative des objets dans un système informatique. DARE utilise la notion de rôle pour modéliser des supports d'activité coopérative. Il serait donc possible que les avancées dans le premier domaine fournissent des solutions au second qui nous intéresse plus particulièrement. Il existe

néanmoins quelques différences à prendre en compte entre la modélisation objet et la modélisation de supports d'activités. Dans la modélisation de l'activité des objets d'un système, les actions et réactions des objets jouant un certain rôle dans une collaboration sont codées dans leurs classes. Dans un support d'activité, les actions et réactions sont aussi celles des sujets qui sont externes au système.

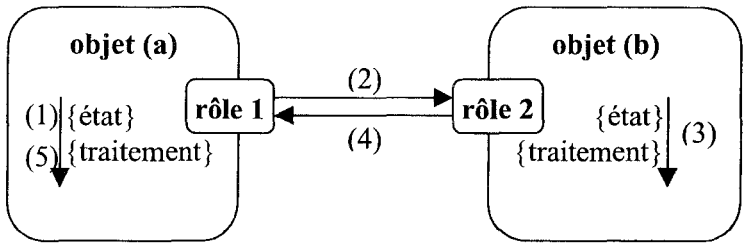

Figure 60. Un exemple d'exécution d'un schéma de collaboration entre deux objets

La Figure 60 présente un exemple de collaboration entre deux objets dans un système. Cette collaboration est déterminée par un schéma introduisant les rôles 1 et 2. Le rôle 1 est joué par l'objet (a) et le rôle 2 par l'objet (b). La classe de l'objet (a) implémente le rôle 1. Ceci signifie que lorsque l'objet (a) atteint un certain état (1), son rôle lui ordonne de déclencher une méthode (2) sur l'objet qui joue le rôle 2. Cette méthode a pour effet un certain traitement (3) interne à l'objet (b) pouvant l'amener à un changement d'état. Ceci amène l'objet (b) qui joue le rôle 2 à déclencher une méthode (4) sur l'objet (a) qui joue le rôle 1. Cette méthode a pour conséquence la suite du traitement (5) des données contenues dans l'objet (a). L'ensemble de ces interaction est déterminée par le schéma de collaboration défini entre les rôles 1 et 2.

La Figure 61 expose une manière de mettre en œuvre un schéma de collaboration au sein de DARE. Dans un souci de lisibilité, nous n'avons pas fait apparaître les micro-rôles. Ainsi, dans cette figure, chaque rôle reçoit des informations et agit sur l'ensemble des outils du sujet dans un support d'activité. La différence majeure avec celle de la Figure 60 se situe au niveau du traitement. En effet, les rôles collaborent de manière à empêcher chaque sujet d'utiliser certaines fonctions de ses outils. Pour ce faire, un rôle agit sur les outils du sujet qui le joue ainsi que sur les rôles des autres sujets impliqués dans la collaboration. Le sujet voit ses outils et perçoit leurs évolutions. Notons que d'autres informations peuvent parvenir au sujet du fait que les applets liées aux outils peuvent être partagées. Grâce à ces informations, un sujet décide d'agir ou non. Il est à priori impossible de prévoir les réactions des sujets. C'est

pourquoi, dans une modélisation par les rôles d'un support à l'activité, la notion de devoir tient une grande place. En effet, à la différence d'un objet dont le comportement est spécifié par un développeur, il est difficile d'obliger un sujet peu motivé à réaliser sa part dans la division du travail. Une fois encore, ceci montre que *dans notre domaine, il est parfois nécessaire de modifier un schéma de collaboration en cours d'exécution pour permettre à l'activité d'être réalisée.*

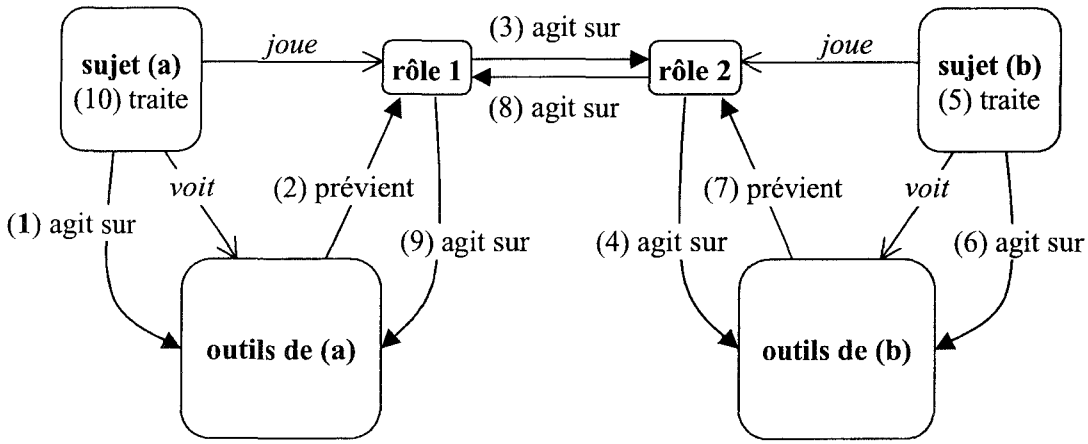

Figure 61. Un exemple d'exécution d'un schéma de collaboration dans un support d'activité

Malgré tout, cet exemple expose une certaine similitude entre la modélisation par les rôles du génie logiciel, et nos problèmes de spécification de rôles dans une tâche. Les mécanismes supportés comme l'héritage-spécialisation, la composition et l'agrégation correspondent à certaines de nos attentes. L'héritage-spécialisation permet de définir de nouveaux schémas de collaboration et de nouveaux rôles à partir d'autres. Par exemple, un schéma de collaboration client-serveur peut devenir un schéma de collaboration particulier entre un maître et son élève. La composition permet par exemple à un sujet de jouer plusieurs micro-rôles. L'agrégation correspond dans le modèle actuel de DARE à un rôle composé d'un agrégat de micro-rôles. Notre objectif est donc de suivre plus avant cette piste de manière à déterminer de manière précise quels sont outils et propriétés des méthodes de modélisation telles OOram qui peuvent apporter des solutions à nos problèmes.

Notons que la modélisation par les rôles, à l'instar de la démarche adoptée dans nos travaux, apparaît comme une nouvelle forme d'utilisation des concepts utilisés dans les sciences humaines pour servir à la conception de systèmes informatiques.

8.5.2 Vers un modèle de composants pour le TCAO

Un autre travail vient d'être amorcé dans le cadre du stage de DEA en informatique de Thomas Vantroys au sein du laboratoire Trigone (2000), travail qui devrait se poursuivre dans le cadre d'une thèse. Ce travail porte sur la définition d'un modèle de composant pour le TCAO. Il découle des problème que nous avons soulevé concernant le modèle des applets. Les applets sont difficilement intégrable dans DARE. Il ne s'agit pas là réellement d'un problème technique (nous avons montré que c'est faisable). Le problème est que l'utilisation que nous en faisons détourne quelque peu les applets du but pour lequel elles ont été crées. Les applets répondent au modèle de composant des JavaBeans. Nous avons constaté que ce modèle est trop général pour nos besoins. Il nous faut donc spécifier un nouveau modèle de composant pour le TCAO. Ce modèle doit non seulement inclure des conventions pour les noms des méthodes et événements 'coopératifs' du composant, mais aussi fournir une description compréhensible par les sujets. Cette description devrait permettre de faire des requêtes sémantiques[1] dans le dépôt de composants. Ils pourront ainsi être liés plus aisément à DARE. D'autre questions viennent se greffer. Faut-il que les composants soient coopératifs, partagés ou isolés (cf. chapitre 7) ? Sont-ils eux-mêmes constitués de composants, comme des micro-rôles déjà livrés avec le composant ?

Ces questions sont pour l'instant sans réponse. Toutefois, les travaux ont déjà été entamés. Ils sont principalement basés sur la mise en œuvre d'Enterprise JavaBeans (EJB) [41] liés à des fichiers de description (notices) en eXtensible Mark-up Language (XML) [84].

8.5.3 Le Meta-Object Facility

Suite à notre réalisation, nous avons également travaillé sur l'introduction du Meta-Object Facility (MOF) [98][26] dans le TCAO [74]. Ce travail est mené en collaboration avec Xavier Lepallec, actuellement en thèse au laboratoire Trigone, et Yvan Peter, Maître De Conférence dans ce même laboratoire. Mais tout d'abord, qu'est-ce que le MOF ?

Après avoir apporté grâce à CORBA une solution technique aux problèmes liés à l'hétérogénéité, l'OMG a travaillé sur les problèmes restant d'actualité en terme d'interopérabilité au niveau sémantique. Le standard MOF [98], paru en 1997, cherche à lever cette limite en permettant l'échange et le partage de modèles. Le partage peut être réalisé dans un mode déconnecté (statique), par transmission d'un fichier, ou dans un mode connecté

(dynamique), en utilisant des objets CORBA. La première approche est la plus répandue et a déjà montré son intérêt dans le cadre de l'interopérabilité entre ateliers de génie logiciel. La deuxième approche reste peu utilisée mais présente un fort potentiel dans la mesure où elle permet une intégration forte avec le monde CORBA et la manipulation dynamique de méta-modèles. Cet accès aux méta-informations sur les modèles offre des perspectives en termes de flexibilité et de généricité. Pour plus de détails à ce niveau, le lecteur pourra se référer à une de nos récentes publications sur ce sujet [74].

Le MOF s'adresse donc à la méta-méta-modélisation. Pour décrire un méta-modèle, il nous faut un modèle de plus haut niveau : un méta-méta-modèle. La Figure 62 expose la place du MOF par rapport à la modélisation et la méta-modélisation. Rappelons que DARE repose sur un méta-modèle permettant de manipuler des modèles. Cette manipulation suppose que les outils de DARE comprennent eux-mêmes ce méta-modèle. Une description de méta-modèle peut être réalisée à partir d'un méta-méta-modèle. Le MOF propose un standard et des outils permettant la manipulation de méta-modèles [72], comme celui de DARE. De plus, les outils du MOF sont liés au monde CORBA, comme ceux de DARE. De ce fait, nous pensons utiliser le MOF dans la suite de nos travaux à plusieurs niveaux.

méta méta modélisation	Meta-Object Facility (MOF)	
méta modélisation	UML	méta modèle DARE
modélisation	modèle UML	modèle DARE

Figure 62. La place du MOF

Pour l'instant, les utilisateurs de DARE redéfinissent les tâches, outils, rôles et micro-rôles à partir de formulaires. A l'avenir, DARE devrait permettre une manipulation graphique de ces éléments. Le MOF permettant d'intégrer tout type de modèles, l'apparition de modeleurs graphiques génériques, s'adaptant grâce à une définition MOF d'un méta-modèle, est fort probable. Ce type d'outil allégerait grandement, et rendrait générique le travail nécessaire à la conception d'un modeleur graphique pour DARE. Ainsi, par exemple, si notre modèle de rôle évolue, il entraînera une altération du méta-modèle de DARE. Si l'outil est compatible MOF et que le nouveau méta-modèle est décrit en MOF, alors l'outil peut charger dynamiquement

[1] au niveau d'abstraction de l'activité

le nouveau méta-modèle et permettre directement, sans modification ou recompilation, la génération ou modification de modèles de supports d'activité.

De manière similaire, l'apparition de services de persistances génériques faciliterait la mise en place, dans DARE, d'un dépôt de modèles. Il s'agit en fait du dépôt de composants utilisés et créés dans DARE. Ainsi, le MOF sera utilisé pour stocker les informations et méta-informations concernant des composants contenant l'héritage historique et culturel de leur utilisation. Ces méta-informations pourront être utilisées pour classer ou effectuer des recherches sur les composants.

Un des objectifs, à long terme, est de favoriser l'interopérabilité entre plusieurs serveurs DARE, ou même d'autre systèmes informatiques. Dans ce deuxième cas de figure les informations concernant un membre dans un système informatique (son nom, ses droits, etc.), pourraient par exemple être utilisées au sein d'un serveur d'activités. Le MOF établissant un protocole standard pour accéder à des méta-objets, il facilitera le travail de mise en place de liens entre un serveur DARE et tout autre système compatible MOF.

Pour conclure, ces types d'utilisation du MOF dans DARE sont très prometteurs, à la fois pour le TCAO (interopérabilité entre collecticiels, etc.), et pour l'avenir du MOF. En effet, une des grandes questions à l'heure actuelle à son sujet est : à quoi peut-il bien servir ? Nous pensons fournir un exemple concret.

8.6 Apports pour l'Apprentissage Coopératif Assisté par Ordinateur (ACAO)

Nous avons présenté dans le chapitre 1 comment le méta-collecticiel ODESCA a été utilisé dans le domaine de l'Apprentissage Coopératif Assisté par Ordinateur (ACAO). De la même manière, les résultats de nos travaux sont destiné à être utilisé dans ce domaine. En particulier, DARE est plus qu'un méta-collecticiel : c'est un collecticiel réflexif. Le nouveau processus de conception d'environnements de TCAO que nous avons proposé peut s'appliquer à l'ACAO en apportant d'ailleurs des bénéfices particuliers à ce domaine particulier. Les propriétés de DARE permettant l'auto organisation au sein d'un groupe d'apprenants ainsi que la co-construction de leur activité collaborative d'apprentissage [24] est du point de vue de l'ACAO une importante contribution. Ceci s'avérerait réellement de valeur en favorisant le développement des capacités méta-cognitives des apprenants [119].

Ainsi, dans le domaine de l'ACAO, les résultats de nos travaux constituent les bases à la réponse faite par le laboratoire Trigone et son partenaire la société Archimed à l'appel à propositions RNTL 2000. Ce projet est nommé « Une Approche Composants des Environnements Virtuels d'Apprentissage » (ACEVA). Comme son nom l'indique, il s'agit principalement de réaliser un environnement d'ACAO basé sur une approche composants intégrés dans une architecture évolutive basée sur Internet et fondée sur la Théorie de l'Activité, faisant de celui-ci un environnement malléable. Ces principes sont ceux que nous avons développés dans le cadre de cette thèse. Ce projet bénéficiera d'interactions avec le projet européen DiViLab [27], un projet IST dont notre laboratoire assume la direction scientifique. DiViLab a pour objectif le développement d'un laboratoire virtuel distribué pour l'enseignement des sciences. A ce titre, DiViLab doit produire des composants éducatifs qui pourront être utilisés dans DARE ou ACEVA. D'autre part, l'expérience dans la mise en œuvre de ces composants pourra nourrir les réflexions menées dans la cadre de DiViLab.

8.7 Les objectifs à plus long terme : une question de temps

Enfin, parallèlement à tous ces travaux, notre objectifs est d'étudier encore plus avant les théories des sciences humaines et celles de l'informatique de manière à faire évoluer le modèle et méta-modèle de la tâche. En particulier, et comme nous l'avons souligné, il nous faut maintenant prendre en compte les contraintes pouvant exister au sein d'une tâche, ainsi que celles pouvant être spécifiées entre des tâches différentes. Ces travaux nous amèneront à introduire plus explicitement la notion du temps dans DARE. Il semble en effet que le temps soit un élément important du problème car nous avons insisté sur le fait qu'un environnement de TCAO doit permettre de cristalliser et réutiliser l'expérience des sujets. Or, pour que l'expérience se crée, il faut du temps. Dans la théorie de l'activité, on parle d'ailleurs d'héritage historique. De ce fait, nous pensons que la notion du temps prendra à l'avenir une place de plus en plus importante dans nos travaux. Néanmoins, ces éléments soulignent un certain paradoxe dans un système comme DARE.

L'activité possède une tendance à continuellement modifier les éléments qui la constituent. Chaque modification est le résultat d'une certaine expérience. Cependant, l'expérience ne peut se construire que pendant une période de stabilité. Par exemple, pour que les sujets puissent utiliser correctement le système, il faut qu'ils en comprennent les mécanismes. Pour réaliser correctement une activité coopérative, un sujet doit comprendre son rôle, ce qui est à

priori impossible si ce rôle ne cesse d'être modifié. La question est alors, combien de temps les éléments qui constituent l'activité doivent-ils rester stables avant une nouvelle évolution ? En d'autres termes, à partir de quel moment une nouvelle évolution est plus susceptible d'aider la réalisation de l'activité, plutôt que de la perturber ?

Ces problèmes, comme de nombreux autres, sont encore aujourd'hui ouverts. Toutefois, nous espérons que l'utilisation de DARE, impliquant des sujets dans de réelles activités coopératives, nous permettra de découvrir les éléments de réponse à ceux qui motivent nos travaux.

Conclusion

Nous avons présenté dans le chapitre huit le bilan de nos travaux et les perspectives envisagées pour nos recherches dans le domaine s'intéressant au Travail Coopératif Assisté par Ordinateur. Nous aimerions cependant revenir dans cette conclusion sur la démarche générale que nous avons adoptée au cours de cette thèse.

De manière à servir la conception d'une application générique dédiée à un domaine spécifique, nous avons tenté de rapprocher les fondements des sciences humaines de ceux de l'informatique. Dans une démarche similaire, la Technométhodologie proposée par Dourish et Button (présentée dans le chapitre 2) tente d'introduire les apports de l'éthnométhodologie dans la conception des outils informatiques en général. A plusieurs reprises (chapitres 5 et 8), nous avons mentionné une certaine approche de l'ingénierie logicielle qui introduit la modélisation par les rôles. Plus récemment encore, Smordal [209] a proposé d'utiliser la Théorie de l'Activité pour classifier les différentes méthodes de modélisation de l'ingénierie logicielle en fonction du but du système qui doit être modélisé. Dans des domaines encore différents, on trouve par exemple la recherche sur les systèmes d'agents qui s'intéresse à la cognition, les algorithmes génétiques...

Notre constat est qu'il existe de plus en plus d'approches dans la recherche en informatique, même fondamentale, qui s'inspirent de disciplines non informatiques pour atteindre leurs objectifs. Ces nombreux exemples, et notre propre expérience, nous laissent à penser qu'il est réellement bénéfique à la recherche en informatique que celle-ci soit de moins en moins isolée

et de mieux en mieux comprise. Nous sommes persuadés que pour aller plus avant, il est nécessaire pour le chercheur de conserver sa spécificité et sa spécialité tout en sachant ne pas s'isoler dans sa discipline. A l'image de celui initié entre Dourish et Button, il doit exister un réel dialogue entre les scientifiques impliqués dans des domaines de recherche différents mais favorables à des interactions potentielles. Nous pensons que l'objectif à atteindre est que chacun apporte son expérience et augmente son propre savoir au contact de l'autre.

L'époque actuelle semble d'ailleurs favorable à l'essor de tels échanges, en particulier grâce à des supports comme l'Internet, ou encore les résultats attendus de notre domaine particulier de recherche, le TCAO. Il est intéressant de constater une certaine convergence entre les conclusions que nous avons formulées dans nos travaux, et la vision de l'Internet du futur prônée par le Réseau National de Recherche en Télécommunications (RNRT) [113]. Dans cette thèse, nous avons proposé un nouveau processus de conception pour les environnements de TCAO, plaçant des sujets d'expertises complémentaires au même plan par rapport à un environnement de travail alors en continuelle co-construction. De la même manière, le rapport sur l'Internet du futur insiste fortement sur la place égale aujourd'hui tenue par le producteur et le consommateur d'information, du point de vue des technologies pour les services avancés d'intermédiation. Dans cette nouvelle répartition, le producteur est aussi consommateur, et vice versa. Dans un processus similaire, et grâce à de tels supports, nous pensons que le chercheur dans un domaine particulier sera de plus en plus consommateur d'informations produites au sein d'un autre domaine de recherche, et producteur d'un certain savoir non seulement fondamental pour sa discipline, mais aussi interdisciplinaire et co-construit.

Dans le cadre de cette thèse, nous avons largement utilisé des informations tirées de la Théorie de l'Activité. Cette théorie nous a permis d'étendre notre point de vue sur des concepts fondamentaux de l'informatique d'aujourd'hui comme les composants, les architectures cadres, ou la méta-modélisation. Un de nos buts, en dehors de la poursuite de nos travaux dans le domaine particulier du TCAO, est d'étudier plus en profondeur les fondements de l'informatique à la lumière de ceux de l'activité humaine. En effet, au-delà du TCAO, le but de l'informatique est bien de supporter une partie de l'activité de ses utilisateurs. Nous pensons qu'une nouvelle approche s'impose car ces utilisateurs, en grande majorité, ne sont plus aujourd'hui des spécialistes ou des informaticiens.

Index

M

N

O

P

R

S

T

U

V

W

Références

1. Bardram J. E., Plans as Situated Action : An Activity Theory Approach to Workflow Systems, Proceedings of the Fifth European Conference on Computer Supported Cooperative Work, Kluwer Academic Publishers, 1997, pp 17-32
2. Bardram J., Designing for the dynamics of cooperative work activities, Proceedings of the ACM CSCW'98 conference, ACM Press, 1998, pp. 89-98
3. Bedny G., Meister D., *The Russian theory of activity, Current Applications to Design and Learning*, Lawrence Erlbaum Associates Publishers, 1997, 430 p
4. Béguin P., Rabardel P., Designing for instrument mediated activity. A paraître dans "Information technology in human activity", Eds. Bertelsen O., Bodkers S.,
5. Bellamy R. K. E., Designing Educational Technology : Computer-Mediated Change, dans [42], 1996, pp. 123-146
6. Benford S., Prinz W., Mariani J., Rodden T., Navarro L., Bignoli E., Brown C. G., Naslund T., MOCCA - A Distributed Environment for Collaboration, Actes de Telepresence'93, First International Conference in Technologies and Theories for Human Cooperation, Collaboration, Coordination, APPLICA'93, Lille, France, 22-24 Mars 1993, pp. 135-148
7. Benford, S., A Real World Metaphor for a CSCW Environment, CoTech WG2 Internal Document, 1991
8. Bertelsen O. W., Elements to a theory of design artefacts: a contribution to critical systems development research, Ph.D.-Thesis, Aarhus University, DAIMI, 1998, PB-531

9. Bodkers S., Activity Theory as a challenge to system design, dans *"Information systems Research: Contemporary Approaches and Emergent Traditions"*, Nissen H., Klein H., Hirschheim R. (eds), Elesevier Science Publishers, BV (North Holland), 1991, pp. 551-564
10. Bodkers S., Christiansen E., Scenarios as springboards in CSCW design, dans *[12]*, 1997, pp. 217-233
11. Bodkers S., Christiansen E., Thüring M., A conceptual toolbox for designing CSCW applications, dans Proceedings of COOP'95 workshop, INRIA, 1995, pp. 266-284
12. Bowkers G., Leigh Star S., Turner W., Gasser L., *Social science, technical systems and cooperative work: beyond the great divide,* Lawrence Erlbaum Associates, *Computer, cognition and work* series, 1997
13. Button G., Dourish P., Technomethodology: Paradoxes and Possibilities. Proceedings CHI'96, 1996
14. Cerrato T. I., Activité collaborative sur réseau : une approche instrumentale de l'écriture en réseau, Th. de Doctorat en psychologie cognitive, Université de Paris 8, 1999
15. Christiansen E., Tamed by a Rose : Computers as Tools in Human Activity, dans [94], 1996, pp 174-198
16. Codella C., Dillenberg D., Ferguson D., Jackson R., Mikalsen T., Silva-Lepe I., Support for Entreprise JavaBeans in Component Broker. In *IBM, Systems Journal*, vol 37, n°4, 98, 1998, pp 454-474
17. Co-Learn, Projet européen DELTA D2005, 1992-1995
18. COMIC, *Delivrable D1.1, COMIC Esprit basic research project 6225*, October 1st, 1993
19. Davidov V. V., Zinchenko V. P., Talynzina N. F, The problem of activity in the work of A. N. Leont'ev. Soviet Psychology 21(4), 1983, pp 31-42
20. DCOM Technical Overview, MicroSoft Corp, Redmond Wash, 1996
21. De Michelis. G., Dubois E., Jarke M., Mylopoulos J., Schmidt J. W., Woo C., Yu E., A three-faceted view of information systems, Communications of the ACM, December 1998, Vol. 41, n°. 12
22. Derycke A., Hoogstoel F, Some Issues in the Design of the Virtual Campus", FIE'98, Frontiers in Education'98, IEEE, November 4-7, 1998, Tempe, Arizona
23. Derycke A., Integration of the learning processes into the Web: Learning Activity Centred Design and Architecture, Webnet'98 conference, invited conference, Orlando, FL, November 1998

24. Derycke A., Kaye A., Participative modelling and design of collaborative learning tools in the CO-LEARN project, In G. Davis, B. Samways (eds), IFIP, Teleteaching 93 Conference, Trondheim, August 20-25 1993, North-Holland, Amsterdam, pp. 191-200
25. Derycke A., Viéville C., Real-time multimedia conferencing system and collaborative learning, *Collaboration Dialogue Technologies in distance education*, Verdejo, F., Cerri, S. (eds), NATO ASI Series, Springer Verlag, Berlin, 1994, pp. 236-256
26. Distributed Systems Technology Centre. *Meta-Object Facility Information.* http://www.dstc.edu.au/Research/Projects/MOF/
27. DIVILAB, Projet IST-1999-217. http://www.divilab.org/
28. Dourish P., Bellotti V., Awareness and Coordination in Shared Workspaces, Proceedings of CHI'92, Monterey, May 1992
29. Dourish P., Meta-level Architectures and CSCW : Designing for Change, Position Paper for CSCW'92 Tools and Technologies Workshop, Toronto, 1992
30. Dourish P., Developing a Reflective Model of Collaborative Systems, ACM Transaction on Computer-Human Interaction, vol. 2, n°1, 1995, pp 40-63
31. Dourish P., Open Implementation and Flexibility in CSCW Toolkits, Ph.D. Thesis, University College London, 1996
32. Dourish P., Extending Awareness Beyond Synchronous Collaboration, CHI'97, Workshop on Awareness in Collaboration Systems, Atlanta, Georgia, May 1997
33. Dourish P., Rethinking Software Representations, Workshop on Representations in Interactive System Design (London, July), 1997 http://www.parc.xerox.com/csl/members/dourish/papers/representation.html
34. Dourish P., Using Metalevel Techniques in a Flexible Toolkit for CSCW Applications, ACM Transaction on Computer-Human Interaction, vol. 5, n°2, 1998, pp 109-155
35. Dourish P., Button G., On "Technomethodology": foundational relationships between éthnomethodology and system design, *Human-Computer Interaction,* vol. 13, Lawrence Erlbaum Associates, 1998, pp. 395- 432
36. Dourish P., Edwards W., A Tale of Two Toolkits : Relating Infrastructure and Use in Flexible CSCW Toolkits, Computer Supported Cooperative Work, vol 9, 2000, pp 33-51
37. Eckstein et al., *Java Swing*, (Publisher) O'Reilly and Associates. ISBN 1-565-92455-X, 1999
38. Ellis C., Wainer J., A conceptual model of Groupware, dans proceedings of the ACM CSCW'94 conference, ACM Press, 1994, pp 79-88

39. Engeström Y., Brown K., Christopher L., Gregory J., Coordination, cooperation and communication in the courts, dans Cole M., Engeström Y., Vasquez O. (eds), *Mind, Culture and Activity*. Cambridge University Press, Cambridge, UK, 1997
40. Engeström Y., *Learning by expanding: an activity-theoretical approach to developmental research,* Orienta-Konsultit Oy, Helsinki, 1987
41. ENTERPRISE JAVABEANS™ SPECIFICATION 1.1. http://java.sun.com/products/ejb/
42. Evolving use of groupware. A full-day workshop at ECSCW'99, Copenhagen, Denmark., 12 September 1999, http://wwwold.telin.nl/events/ecscw99evo/index.html
43. Fayad M., Schmidt D., Object-Oriented Application Frameworks, Communications of the ACM, 40 (10), 1997, pp 32-38
44. Fitzpatrick G., Kaplan S., Mansfield T., Applying the Locales Framework to Understanding and Designing, Proceedings of Australasian Computer Human Interaction Conference (OzCHI'98), Adelaide Australia, IEEE, 1998, pp 122-129
45. Frankel D., Technology-independent business object – the concept of a meta-model, Java Report, November 1998, pp 71 -78
46. G. Crean, M. O'Sullivan, MODEM: A New European Microelectronics Telematics Based Educational Initiative, dans Proceedings of the 1997 IEEE Int. Conference on Microelectronic Systems Education (MSE '97), Arlington (Virginia), July 21-23 1997
47. Gangopadhyay D., Pree W., Schappert A., Report on the Workshop Framework—Centered Software Development, OOPSLA'95, 1995, pp 100-104
48. Geib J-M., Gransart C., Merle P. *CORBA : Des concepts à la pratique.* Edition InterEditions, 1999
49. Harrap's compact, *Dictionnaire Anglais-Français / Français-Anglais.* Harrap Books Ltd, Part 2, 1992, p 94
50. Hoogstoel F., Bourguin G., Using a WWW server to access and manage a Gemstone Smalltalk server supporting a virtual collaborative learning organisation, Actes sur CD-ROM de la conférence WebNet'96, organisée par l'AACE (Association for Advance Computing in Education) et de la WebSociety, San Francisco, CA, 15-19 Octobre 1996
51. Hoogstoel F., Tarby J-C., Integrating a user interface design method in a CSCW framework, Actes du troisième atelier international sur le Groupware CRIWG'97, Madrid, Espagne, 1 - 3 octobre 1997
52. Hoogstoel F., Une approche organisationnelle du Travail Coopératif Assisté par Ordinateur. Application au projet Co-Learn, Th. de Doctorat en Informatique, Université des Sciences et Technologies de Lille, 1995, n° 1487

53. Hoogstoel F., Vieville C., Pastor E., The AMIGO Approach : the information model, dans [102], 1992, pp59-89.
54. Hummes J., Kohrs A., Merialdo B., Software Components for Co-operation: a Solution for the "Get help" Problem, Proceeding of COOP'98 conference, INRIA, 1998, http://eurecom.fr/~hummes/docs/COOP98/coop98-submitted.html/
55. Hummes J., Merialdo B., Object Components for Cooperation. A Highly Customizable Tutoring-System with JavaBeans, ECSCW'97 OOGP workshop, 1997
56. Hummes J., Merialdo B., Design of extensible component-based groupware, soumis pour être publié par Kluwer dans le Journal of CSCW, 1999, http://www.eurocom.fr/~hummes/docs/JCSCW/JCSCW.html .
57. JavaSoft, JavaBeans 1.0 API Specification, Mountain View, California, SUN Microsystems, 1997
58. Kaasvol J., Ogrim L., Super-users: Hackers, Managements Hostages, or Working-class Heroes. A Study of User Influence on Redesign in Distributed Organizations, Proceedings IRIS-17 Seventeenth Information Systems Resarch Seminar in Scandinavia. (Syote), Dept. Of Information Processing Science, University of Oulu, Finland, 1994, pp 784-798
59. Kaplan S., Fitzpatrick G., Mansfield S., Tolone W., MUDling Through, Proceedings HICSS'97, vol. 2, IEEE Computer Society Press, 1997, pp 539-548.
60. Kaptelinin V., Activity Theory: Implications for Human-Computer Interaction, *dans [94]*, 1996, pp 103-116
61. Kaptelinin V., Computer-Mediated Activity : Functional Organs in Social an Developmental Contexts, dans [94], 1996, pp. 45-68
62. Kaptelinin V., Nardi B,. Macaulay C., The Activity Checklist : a Tool for Representing the "Space" of Context., Journal of Interactions, July-August 1999, pp 27-39
63. Kiczales G., Beyond the black box: open implementation, IEEE Software, January 1996
64. Kiczales G., Bobrow D.G., Des Rivieres J. *The Art of the Metaobject Protocol*, MIT Press, August 1991, 335 p
65. Klöckner K. et al., POLITeam --- Bridging the gap between Bonn and Berlin for and with the users, Proceedings of ECSCW'95, Stockholm, Sweden, 1995, pp 17-32
66. Koch M., Teege G., Support for tailoring CSCW systems: adaptation by composition. Seventh Euromicro Workshop on Prallel an Distributed Processing PPD'99, 1999, pp 146-153
67. Kristensen B., Subject Composition by Roles, OOIS'97 – 1997 international conference on object oriented information systems proceedings, Brisbane, 10-12 november 1997

68. Kuutti K., Activity Theory as a Potential Framework for Human-Computer Interaction Research, *dans [95],* 1996, pp 17-44
69. Kuutti K., The concept of activity as a basic unit of analysis for CSCW research, Proceeding of the second ECSCW'91 conference, Kluwers Academics Publishers, 1991, pp 249-264
70. Kuutti K., Notes on systems supporting "Organisational context" – An activity theory viewpoint, COMIC European project, deliverable D1.1, 1993, pp 101- 117
71. Lave J., Wenger E., *Situated learning: legitimate peripheral participation.* Cambridge University Press, Cambridge, UK, 1991
72. Lemesle R. Meta-modelling and modularity : Comparison between MOF, CDIF & sNets formalisms, OOPSLA'98, 1998
73. Leont'ev A. N., Activity, Consciousness and Personality, Political Publishers, Moscow, 1977
74. Lepallec X., Bourguin G., Peter Y., Gestion de méta-données avec le Meta Object Facility, actes de la conférence OCM'2000, Nantes, 18 mai 2000, pp 101-112
75. Lewis R., Apprendre conjointement : une analyse, quelques expériences et un cadre de travail, Proceedings of Hypermedia et apprentissage (INRP), Poitiers, 1998, pp 11-28
76. Mackay W., Users and Customizables Software : A Co-adaptive Phenomenon, Ph. D. Thesis, Sloan School of Management, Massachussetts Instutute of Technology, Cambridge, MA, 1990
77. MacLean A., Carter K., Lövstrand L., Moran T., User-tailorable Systems: Pressing the Issues with Buttons, Proceedings CHI'90 Human Factors in Computing Systems, ACM Press, Seattle, 1-5 April 1990, pp 175-182
78. Maes P., *Computational Reflection*, Ph.D. Thesis, V.U.B, Brussels, 1987
79. Malone T. W., Lai K., Experiments with Oval : A Radically Tailorable Tool for Cooperative Work, MIT, Mars 1992
80. Malone T., Grant K., Lai K., Rao R., Rosenblitt D., Semi-structured messages are surprisingly useful for computer-supported coordination, ACM Transactions on Office Information Systems, vol. 5, 1987, pp 115-131
81. Mansfield T., Kaplan S., Fitzpatrick G., Phelps T., Fitzpatrick M., Taylor R., Segall B., Hering C., Johnson P., Berry A., Evolving Orbit : a progress report on building locales, Proceedings GROUP'97, 1997
82. Mansfield T., Kaplan S., Fitzpatrick G., Phelps T., Fitzpatrick M., Taylor R., Toward Locales : Supporting collaboration with Orbit, à paraître dans Journal of Information an Software Technology, Elsevier, http://archive.dstc.edu.au/worlds/Papers/abstracts.html

83. Martel C., Ferraris C., De la régulation dans les collecticiels, acte de la conférence IHM'99 - Interaction pour tous, Montpellier, France, 22-26 Novembre 1999
84. Michard Alain, *XML, Langage et Applications*, Edition Eyrolles, 1998
85. MOCCA, Models for CSCW Applications, CoTech WG2 Internal Document, 1991
86. MODEM, Telematics Applications Programme : ET1015. Multimedia Optimisation and Demonstration for Education in Microelectronics (MODEM) http://nmrc.ucc.ie/modem/
87. Morch A., Designing for radical tailorability: coupling artifact and rationale, Knowledge-Based Systems, Vol. 7., n° 4, Butterworth-Heinermann Ltd, 1994
88. Morch A. Three levels of end-user tailoring: customization, integration, and extension, *dans* [90], 1995, pp 41-51
89. Morch A., Evolving a Generic Application into a Domain-oriented Design Environment, Scandivian Journal of Information Systems, 1997
90. Morch A., Method and Tools for Tailoring of Object-oriented Applications: An Evolving Artifacts Approach, part 1, Dr. Scient. Thesis Research Report 241, University of OSLO, Department of Informatics, 1997, pp 1-26
91. Morch A., Mehandjev N., Tailoring as Collaboration : The Mediating Role of Multiple Representations and Application Units, Computer Supported Cooperative Work 9, 2000, pp 75-100
92. Muller P-A., *Modélisation objet avec UML*, Edition Eyrolles, 1997
93. Nardi B. A., Small Matter of Programming : Perspectives on End User Computing, Ma : MIT Press, Cambridge, 1993.
94. Nardi B. A., *Context and consciousness : activity theory and Human-Computer Interaction*. Eds., Cambridge, Ma : MIT Press, 1996
95. Nardi B. A., Studying Context : A Comparison of Activity Theory, Situated Action Models, and Distributed Cognition, *dans [94],* 1996, pp 69-102
96. Nouvelles interfaces pour le travail coopératif, projet TL98092, Programme Télécomunications, CNRS, 1998
97. Object Managment Group, XML Metadata Interchange (XMI), Proposal to the OMG OA&DTF RFP 3: Stream-based Model Interchange Format (SMIF), Joint Submission, 1998, http://www.omg.org/cgi-bin/doc?ad/98-10-05.pdf
98. Object Managment Group, *Meta Object Facility (MOF) Specification (Version 1.3 RTF*, 1999, http://www.omg.org/cgi-bin/doc?ad/99-09-04.pdf
99. Object Managment Group, *Meta Object Facility (MOF) Specification (Version 1.3 RTF), Appendix C : MODL Description of the MOF,* 1999
100. Object Managment Group. http://www.omg.org/omg/background.html

101. OMG, A discussion of the object management architecture, Document technnique, Object Management Group, Janvier 1997

102. Open University, Polytechnical University of Madrid, Télésystèmes, Trigone, Representation models for collaborative educational situations and collaborative learning activities. DELTA Project D2005 Co-Learn deliverable, juin 1992

103. Paepcke A., *Object Oriented Programming. The CLOS Perspective*, MIT Presss, Cambridge, Massachussets, London, England, 1993, 353 p

104. Pankoke-Batatz Uta. (eds), Computer Based Group Communication Model, The AMIGO activity model, Ellis Horwood Books in Information Technology, 1989

105. Pree W., Meta patterns-A means for capturing the essentials of reusable Object-oriented desing, actes de ECOOP'94, 1994

106. Pree W., Component-based software Development – A new paradigm in software engineering ?, Software-Concepts and Tools, Springer-Verlag, n° 18, 1997, pp 169-174

107. Prinz W., Mark G., Pankoke-Bataz U., Designing groupware for congruency in use, proceedings of CSCW'98, ACM, 1998, pp 373-382

108. Raeithel A., Velichkovsky. B. M., Joint Attention and Co-Construction: New Ways to Foster User-Designer Collaboration, dans [94], 1996, pp 199-233

109. Reenskaug T., Wold P., Lehne O., Working with Objects. The OOram Software Engineering Method, Mannig Publications Co, 1996, 366 p

110. Reenskaug T., Lehne O., Designing Distributed Systems for Reuse, OOPSLA'97 Tutorial 26, 1997, http://www.ifi. uio.no/~trygver/documents/9710-OOPSLA/sec1.htm

111. Reenskaug T., UML Collaboration semantics. Agreen (?) paper, http://www.ifi.uio.no/~trygver/documents/991108-UML/uml-collaboration.html.

112. Riehle D., Gross T., Role Model Based Framework Design and Integration, OOPSLA'98, Vancouver, ACM, 1998

113. RNRT 2000, projet du Réseau National de Recherche en Télécommunications, 2000, http://www.telecom.gouv.fr/rnrt/

114. Roschelle J., Kaput J., Stroup W. M., Kahn T., Scalable Integration of educational software : exploring the Promise of Component Architecture, http://www-jiime.open.ac.uk/98/6/roschelle-01.html

115. Rumbaugh J. et al., *OMT. Modélisation et conception orientées objet.* Edition française revue et augmentée, Masson, Paris, Prentice Hall, Londre, 1995, p 15

116. Salomon G., *Distributed Cognition*, Cambridge University Press, Cambridge, UK, 1993

117. Sandkuhl K., Nentwig L., Manhart S., Lafrenz P., Redesigning CSCW-System for Network Computing, Experience from the HotCon Project, IEEE, 1998

118. Schuler A., Namioka A., (eds), Participatory Design : Principles and Practice, Lawrence Erlbaum, Hillsdale, NJ, 1993

119. Sherry L. Issues in distance learning, dans *International Journal of Educational Telecommunication*, 1 (4), 1995, pp 337-365

120. Smordal O., Classifying Approaches to Object Oriented Analysis of Work with Activity Theory, OOIS'97 – 1997 international conference on object oriented information systems proceedings, Brisbane, 10-12 november 1997

121. Sommerville I., Bentley R., Rodden T., Sawyer P., Cooperative systems design, dans The Computer Journal, vol. 37, n°5, 1994, pp 357-366

122. Stiemerling O., Supporting tailorability of groupware through component architectures, ECSCW'97 Workshop, 1997

123. Stiemerling O., Cremers A., Tailorable component architectures for CSCW-systems, Proceedings of the 6th Euromicro Workshop on Parallel and Distributed Programming, Madrid, Spain, IEEE Press, Jan 21-24, 1998, pp. 302-308.

124. Stiemerling O., Hinken R., Cremers A., Distributed component-based tailorability for CSCW applications, proceedings of The fourth international symposium on autonomous decentralized systems – integration of heterogeneous systems, Tokyo, IEEE, 1999, pp 345-352

125. Stiemerling O., Hinken R., Cremers A., The EVOLVE tailoring platform: supporting the evolution of component-based groupware, IEEE, 1999, pp 106-115

126. Suchman L., *Plans and Situated Actions,* Cambridge University Press, Cambridge, UK, 1987

127. Sussman D., Homer A., ASP 2.0 Programmer's Reference, Wrox Press Inc, ISBN: 1861002459, 1999

128. Szyperski C., Pfister C., Component-oriented programming: WCOP'96 Workshop Report, Special issues in object-oriented programming: Workshop reader of the 10th European Conference on Object-Oriented Programming ECOOP'96, Linz, 1996, pp 127-130

129. Tolone W. J., Introspect : A meta-level specification framework for dynamic, evolvable collaboration support, Ph. D. Thesis, University of Illinois, 1996

130. Tolone W. J., Kaplan S., Fitzpatrick G., Specifying Dynamic Support for Collaborative Work within wOrlds, proceedings of the 1995 ACM Conference on Organisational Computing Systems (COOCS'95), 1995, pp 55-65

131. Viéville C., An Asynchronous Collaborative Learning System on the Web, *The Electronic University*, Springer Verlag, the CSCW series, 1998, pp 99-113

132. Viéville C., Derycke A., Self Organised Group Activities Supported by Asynchronous Structured Conversations, Proceedings of the IFIP conference on "*Virtual Campus: trends for higher education, and training*", Madrid, Spain, November 1997, F. Verdjo, G.Davies (eds), Chapmann & Hall, London, 1998, pp 191-204

133. Vogel A., Duddy K., Java Programming with CORBA, John Wiley & Sons, Inc, 1997

134. Zinchenko V. P., Developing Activity Theory : The Zone of Proximal Development and Beyond, dans [42], 1996, pp 283-324

Table des figures

Liste des tableaux

Résumé :

Dans le domaine de l'informatique, la recherche sur le Travail Coopératif Assisté par Ordinateur (TCAO) s'interroge toujours sur la démarche de conception et les fondements à adopter pour enfin créer des collecticiels répondant aux besoins des utilisateurs. Les études empiriques tendent à montrer l'invalidité d'une démarche classique de conception qui tente de prévoir à l'avance l'activité particulière d'un groupe de travail pour l'implémenter. En effet, le collecticiel doit pouvoir prendre en compte les besoins émergeants du groupe au fur et à mesure de l'activité.

Dans nos travaux, nous avons adopté une nouvelle démarche pour la conception des environnements de TCAO. Notre objectif n'est plus de comprendre une activité particulière pour la modéliser. Nous voulons comprendre ce qu'est l'activité d'une manière générique, et identifier les concepts et mécanismes fondamentaux qui la caractérisent de manière à directement les rapprocher des fondements informatiques utiles à la conception.

L'étude de la Théorie de l'Activité (AT) nous a permis de spécifier une certaine vision des collecticiels, et d'en proposer une réalisation nommée DARE (Activités Distribuées dans un Environnement Réflexif). Cette réalisation offre à la fois un support à l'activité coopérative, ainsi qu'un support à la méta-activité de redéfinition de l'environnement de travail informatisé. Ainsi dans DARE, les acteurs sont à la fois les utilisateurs et les concepteurs d'un environnement de TCAO qui supporte sa propre activité coopérative de (re)conception.

En nous inspirant directement de l'AT, DARE a été conçu comme un système malléable et distribué (CORBA) sur l'Internet. Il met en œuvre un modèle générique d'activité, ainsi qu'un méta-modèle permettant sa manipulation à un niveau d'abstraction plus proche du domaine. DARE utilise les propriétés réflexives de certains langages (Smalltalk et Java) pour proposer une implémentation ouverte liée à une approche composants.